Walt Truszkowski, Lou Hallock, Christopher Rouff, Jay Karlin, James Rash, Michael G. Hinchey, Roy Sterritt

Autonomous and Autonomic Systems

with Applications to NASA Intelligent Spacecraft Operations and Exploration Systems

July 22, 2009

Springer
Berlin Heidelberg New York
Hong Kong London
Milan Paris Tokyo

Preface

Over the years, NASA missions have been gradually adding autonomy to flight and ground systems to increase the amount of science data returned from missions, perform new science, and reduce mission costs. In the early 1990s, a group of researchers at NASA Goddard Space Flight Center began researching artificial intelligence techniques for application to ground control systems and spacecraft. The research started by developing and experimenting with expert systems to automate ground system software and reduce the number of people needed to control a spacecraft. This was followed by research into agent-based technologies and autonomic systems concepts directed towards enhancing future space-mission self-management and survivability in the harsh environments in which they operate.

This book describes the results of this research, primarily in the context of uncrewed assets or assets that must be capable of operating in untended mode. In addition, it aims to identify and analyze the software concepts needed for greater degrees of autonomy and autonomicity in future NASA space missions. This book is intended to at least partly document NASA's efforts to design software embodying such concepts. Not covered in this book is the supporting hardware of future missions and the intricate software that implements orbit and attitude determination and on-board resource allocation or planning and scheduling (though we refer to these technologies and give references for the interested reader).

The book is divided into three parts: Background, Technologies, and Applications.

In Part One of the book, Chapter 1 gives an introduction to autonomous and autonomic systems, how we define them in this book, and why they are needed in future space missions. Chapter 2 provides an overview of ground and flight software and the functions that each supports. Chapter 3 discusses the reasons for flight autonomy and its evolution over the past 30 plus years. Chapter 4 mirrors Chapter 3 for ground systems.

In Part Two, Chapter 5 goes over the core technologies needed to develop autonomous and autonomic space missions, such as planners, collaborative

languages, reasoners, learning technologies, perception technologies, and verification and validation methods for these technologies. Chapter 6 discusses the design of autonomous spacecraft from an agent-oriented perspective. It covers the idea of a flight software backbone and the spacecraft functions that this backbone will need to support, and the concept of designing a spacecraft as an interacting agent. Chapter 7 covers the technologies needed for cooperative spacecraft. It starts by discussing the need for cooperative spacecraft, a model of cooperative autonomy, mission management issues for cooperation, and core technologies for cooperative spacecraft. Chapter 8 describes autonomic systems and what makes a system autonomic, why autonomicity is needed for future autonomous systems, and what functions would be needed to make future missions autonomic.

Part Three starts with Chapter 9, discussing spacecraft constellations, cooperation between or among the spacecraft in the constellation, difficulties in controlling multiple cooperative spacecraft, and a multi-agent paradigm for constellations. Chapter 10 gives an overview of swarm technology and discusses its application to certain proposed missions, as well as issues that would be expected to arise in developing software for swarm-based missions. Chapter 11 presents some examples of possible future NASA missions that are in the planning or concept-development phase. This chapter examines the possible application of technologies discussed earlier in the book to these new missions and describes additional technologies that will need to be developed to enable these missions to be flown.

The Appendices provide additional information that might be of interest to readers who desire greater knowledge of spacecraft attitude and orbit determination and control. It also includes some operational scenarios representing agent communication between the ground and flight software, as well as a list of acronyms and a glossary of terms. References are provided at the end of the book. The references are representative of the work covered and not exhaustive.

There are four groups of readers who may benefit from this book. First are those who have an interest in spacecraft and desire an overview of ground and spaceflight systems and the direction of related technologies. The second group are those who have a background in developing current flight or ground systems and desire an overview of the role that autonomy and autonomic systems may play in future missions. A third group comprises those who are familiar with autonomous and/or autonomic technologies and are interested in applying them to ground and space systems. A fourth group would be scientists who are looking for ways to do new science. This book can help this group with understanding the technology that can help them achieve their goals.

Different readers in each of the above groups may already have some of the background covered in this book and may choose to skip some of the chapters. Those in the first group will want to read the entire book. Those in the second group could skip Chapter 2 as well as Chapters 3 and 4, though the latter two

may be found interesting from an historical point of view. The third group of readers could skip or skim Chapter 5, and although they may already be familiar with the technologies discussed in Chapters 6, 7, and 8, they may find the chapters of interest relative to the application of AI technologies in the space flight domain.

We hope that this book will not only give the reader useful background information on some of the technologies needed to develop future autonomous and autonomic space missions, but also impart insights into technology gaps relevant to advanced mission concepts and stimulate new ideas and research into enabling technologies for future missions.

Walt Truszkowski
Lou Hallock
Christopher Rouff
Jay Karlin
James Rash
Mike Hinchey
Roy Sterritt

Acknowledgements

There are a number of people who made this book possible. We would like to thank them for their contributions. We have made liberal use of their work and their contributions to Agent-based research at NASA Goddard.

We would like to thank the following people from NASA Goddard Space Flight Center who have contributed information on or some writings to material that covered flight software: Joe Hennessy, Elaine Shell, Dave McComas, Barbara Scott, Bruce Love, Gary Welter, Mike Tilley, Dan Berry and Glenn Cammarata.

We are also grateful to Dr. George Hagerman (Virginia Tech) for information on Virginia Tech's Self Sustaining Planetary Exploration Concept; to Dr. Walter Cedeño (Penn State University, Great Valley) for information on Swarm Intelligence; Steve Tompkins (NASA Goddard) for information on future NASA missions, to Drs. Jonathan Sprinkle and Mike Eklund (University of California at Berkeley) for information on research into UUVs being conducted at that institution, to Dr. Subrata Das and his colleagues at Charles River Analytics of Boston, for their contribution of information concerning AI technologies, and to Dr. Jide Odubiyi for his support of Goddard's agent research, including his major contribution to the development of the AFLOAT system.

We would like to thank the NASA Goddard Agents Group for their work on the Agent Concept Testbed (ACT) demonstration system and their work on spacecraft constellations and the NASA Space Operations Management Office (SOMO) Program for providing the funding. We would also like to thank Drs. Mike Riley, Steve Curtis, Pam Clark, and Cynthia Cheung of the ANTS project for their insights into multi-agent systems.

The work on autonomic systems was partially supported at the University of Ulster by the Computer Science Research Institute (CSRI) and the Centre for Software Process Technologies (CSPT), which is funded by Invest NI through the Centres of Excellence Programme, under the EU Peace II initiative.

The work on swarms has been supported by the NASA Office of Systems and Mission Assurance (OSMA) through its Software Assurance Research Program (SARP) project, Formal Approaches to Swarm Technologies (FAST), administered by the NASA IV&V Facility, and by NASA Goddard Space Flight Center, Software Engineering Laboratory (Code 581).

Contents

Part I Background

1 Introduction . . . 3
1.1 Direction of New Space Missions . . . 5
1.2 Automation vs. Autonomy vs. Autonomic Systems . . . 9
1.2.1 Autonomy vs. Automation . . . 9
1.2.2 Autonomicity vs. Autonomy . . . 11
1.3 Using Autonomy to Reduce the Cost of Missions . . . 14
1.3.1 Multi-Spacecraft Missions . . . 14
1.3.2 Communications Delays . . . 16
1.3.3 Interaction of Spacecraft . . . 16
1.3.4 Adjustable and Mixed Autonomy . . . 17
1.4 Agent Technologies . . . 17
1.4.1 Software Agents . . . 19
1.4.2 Robotics . . . 22
1.4.3 Immobots or Immobile Robots . . . 23
1.5 Summary . . . 23

2 Overview of Flight and Ground Software . . . 25
2.1 Ground System Software . . . 25
2.1.1 Planning and Scheduling . . . 27
2.1.2 Command Loading . . . 28
2.1.3 Science Schedule Execution . . . 28
2.1.4 Science Support Activity Execution . . . 28
2.1.5 Onboard Engineering Support Activities . . . 28
2.1.6 Downlinked Data Capture . . . 29
2.1.7 Performance Monitoring . . . 29
2.1.8 Fault Diagnosis . . . 29
2.1.9 Fault Correction . . . 30
2.1.10 Downlinked Data Archiving . . . 30
2.1.11 Engineering Data Analysis/Calibration . . . 31

2.1.12 Science Data Processing/Calibration 31
2.2 Flight Software 31
2.2.1 Attitude determination and control, Sensor Calibration, Orbit Determination, Propulsion 33
2.2.2 Executive and Task Management, Time Management, Command Processing, Engineering and Science Data Storage and Handling, Communications 34
2.2.3 Electrical Power Management, Thermal Management SI Commanding, SI Data Processing 34
2.2.4 Data Monitoring, Fault Detection and Correction 35
2.2.5 Safemode 35
2.3 Flight vs. Ground Implementation 35

3 Flight Autonomy Evolution 37
3.1 Reasons for Flight Autonomy 38
3.1.1 Satisfying Mission Objectives 39
3.1.2 Satisfying Spacecraft Infrastructure Needs 47
3.1.3 Satisfying Operations Staff needs 51
3.2 Brief History of Existing Flight Autonomy Capabilities 55
3.2.1 1970s and Prior Spacecraft 55
3.2.2 1980s Spacecraft 57
3.2.3 1990s Spacecraft 59
3.2.4 Current Spacecraft 62
3.2.5 Flight Autonomy Capabilities of the Future 63
3.3 Current Levels of Flight Automation/Autonomy 66

4 Ground Autonomy Evolution 69
4.1 Agent-based Flight Operations Associate 69
4.1.1 A Basic Agent Model in AFLOAT 70
4.1.2 Implementation Architecture for AFLOAT Prototype 74
4.1.3 The Human Computer Interface in AFLOAT 75
4.1.4 Inter-agent Communications in AFLOAT 76
4.2 Lights Out Ground Operations System 79
4.2.1 The LOGOS Architecture 79
4.2.2 An Example Scenario 81
4.3 Agent Concept Testbed 81
4.3.1 Overview of the ACT Agent Architecture 82
4.3.2 Architecture Components 83
4.3.3 Dataflow Between Components 87
4.3.4 ACT Operational Scenario 89
4.3.5 Verification & Correctness 91

Part II Technology

5 Core Technologies for Developing Autonomous and Autonomic Systems 95
5.1 Plan Technologies 96
5.1.1 Planners 96
5.2 Collaborative Languages 103
5.3 Reasoning with Partial Information 104
5.3.1 Fuzzy Logic 104
5.3.2 Bayesian Reasoning 105
5.4 Learning Technologies 106
5.4.1 Artificial Neural Networks 106
5.4.2 Genetic Algorithms and Programming 107
5.5 Act Technologies 107
5.6 Perception Technologies 108
5.6.1 Sensing 109
5.6.2 Image and Signal Processing 109
5.6.3 Data Fusion 110
5.7 Testing Technologies 110
5.7.1 Software Simulation Environments 111
5.7.2 Simulation Libraries 113
5.7.3 Simulation Servers 113
5.7.4 Networked Simulation Environments 113

6 Agent-based Spacecraft Autonomy Design Concepts 115
6.1 High Level Design Features 115
6.1.1 Safemode 116
6.1.2 Inertial Fixed Pointing 116
6.1.3 Ground Commanded Slewing 117
6.1.4 Ground Commanded Thruster Firing 117
6.1.5 Electrical Power Management 118
6.1.6 Thermal Management 118
6.1.7 Health and Safety Communications 118
6.1.8 Basic Fault Detection and Correction 118
6.1.9 Diagnostic Science Instrument Commanding 119
6.1.10 Engineering Data Storage 119
6.2 Remote Agent Functionality 119
6.2.1 Fine Attitude Determination 120
6.2.2 Attitude Sensor/Actuator and Science Instrument Calibration 121
6.2.3 Attitude Control 121
6.2.4 Orbit Maneuvering 122
6.2.5 Data Monitoring and Trending 122
6.2.6 "Smart" Fault Detection, Diagnosis, Isolation, and Correction 122
6.2.7 Look-ahead Modeling 123
6.2.8 Target Planning and Scheduling 123

6.2.9 Science Instrument Commanding and Configuration ... 124
6.2.10 Science Instrument Data Storage and Communications . 124
6.2.11 Science Instrument Data Processing ... 124
6.3 Spacecraft Enabling technologies ... 125
6.3.1 Modern CCD Star Trackers ... 125
6.3.2 Onboard Orbit Determination ... 125
6.3.3 Advanced Flight Processor ... 126
6.3.4 Cheap Onboard Mass Storage Devices ... 126
6.3.5 Advanced Operating System ... 126
6.3.6 Decoupling of Scheduling from Communications ... 126
6.3.7 Onboard data Trending and Analysis ... 127
6.3.8 Efficient Algorithms for Look-ahead Modeling ... 127
6.4 AI Enabling Methodologies ... 127
6.4.1 Operations Enabled by Remote Agent Design ... 127
6.4.2 Dynamic Schedule Adjustment Driven by Calibration Status ... 128
6.4.3 Target of Opportunity Scheduling Driven by Realtime Science Observations ... 129
6.4.4 Goal-driven Target Scheduling ... 130
6.4.5 Opportunistic Science and Calibration Scheduling ... 131
6.4.6 Scheduling Goals Adjustment Driven by Anomaly Response ... 131
6.4.7 Adaptable Scheduling Goals and Procedures ... 132
6.4.8 Science Instrument Direction of Spacecraft Operation ... 132
6.4.9 Beacon Mode Communication ... 133
6.4.10 Resource Management ... 134
6.5 Advantages of Remote Agent Design ... 134
6.5.1 Efficiency Improvement ... 134
6.5.2 Reduced FSW Development Costs ... 137
6.6 Mission Types for Remote Agents ... 138
6.6.1 LEO Celestial Pointers ... 139
6.6.2 GEO Celestial Pointers ... 141
6.6.3 GEO Earth Pointers ... 141
6.6.4 Survey missions ... 142
6.6.5 Lagrange Point Celestial Pointers ... 142
6.6.6 Deep Space Missions ... 144
6.6.7 Spacecraft Constellations ... 144
6.6.8 Spacecraft as Agents ... 144

7 Cooperative Autonomy ... 147
7.1 Need for Cooperative Autonomy in Space Missions ... 148
7.1.1 Quantities of Science Data ... 148
7.1.2 Complexity of Scientific Instruments ... 148
7.1.3 Increased Number of Spacecraft ... 148
7.2 General Model of Cooperative Autonomy ... 149

7.2.1 Autonomous Agents . . . 149
7.2.2 Agent Cooperation . . . 151
7.2.3 Cooperative Actions . . . 155
7.3 Spacecraft Mission Management . . . 156
7.3.1 Science Planning . . . 158
7.3.2 Mission Planning . . . 158
7.3.3 Sequence Planning . . . 158
7.3.4 Command Sequencer . . . 158
7.3.5 Science Data Processing . . . 159
7.4 Spacecraft Mission Viewed as Cooperative Autonomy . . . 159
7.4.1 Expanded Spacecraft Mission Model . . . 159
7.4.2 Analysis of Spacecraft Mission Model . . . 162
7.4.3 Improvements to Spacecraft Mission Execution . . . 163
7.5 An Example of Cooperative Autonomy: Virtual Platform . . . 165
7.5.1 Virtual Platforms under Current Environment . . . 166
7.5.2 Virtual Platforms with Advanced Automation . . . 166
7.6 Examples of Cooperative Autonomy . . . 168
7.6.1 The Mobile Robot Laboratory at Georgia Tech . . . 169
7.6.2 Cooperative Distributed Problem Solving Research Group at the University of Maine . . . 170
7.6.3 Knowledge Sharing Effort . . . 171
7.6.4 DIS and HLA . . . 171
7.6.5 IBM Aglets . . . 172

8 Autonomic Systems . . . 175
8.1 Overview of Autonomic Systems . . . 175
8.1.1 What are Autonomic Systems? . . . 176
8.1.2 Autonomic Properties . . . 177
8.1.3 Necessary Constructs . . . 179
8.1.4 Evolution versus Revolution . . . 180
8.1.5 Further Reading . . . 180
8.2 State of the Art Research . . . 182
8.2.1 Machine Design . . . 182
8.2.2 Prediction and Optimization . . . 183
8.2.3 Knowledge Capture and Representation . . . 183
8.2.4 Monitoring and Root-Cause Analysis . . . 183
8.2.5 Legacy Systems and Autonomic Environments . . . 184
8.2.6 Space Systems . . . 185
8.2.7 Agents for Autonomic Systems . . . 185
8.2.8 Policy Based Management . . . 185
8.2.9 Related Initiatives . . . 186
8.2.10 Related Paradigms . . . 186
8.3 Research and Technology Transfer Issues . . . 187

Part III Applications

9 Autonomy in Spacecraft Constellations 191
9.1 Introduction 191
9.2 Constellations Overview 194
9.3 Advantages of Constellations 195
9.3.1 Cost Savings 196
9.3.2 Coordinated Science 196
9.4 Applying Autonomy and Autonomicity to Constellations 197
9.4.1 Ground-based Constellation Autonomy 197
9.4.2 Space-based Autonomy for Constellations 198
9.4.3 Autonomicity in Constellations 199
9.5 Intelligent Agents in Space Constellations 201
9.5.1 Levels of Intelligence in Spacecraft Agents 202
9.5.2 Multi-Agent Based Organizations for Satellites 204
9.6 Grand View 205
9.6.1 Agent development 206
9.6.2 Ground-based autonomy 208
9.6.3 Space-based autonomy 208

10 Swarms in Space Missions 211
10.1 Introduction to Swarms 212
10.2 Swarm Technologies at NASA 213
10.2.1 SMART 214
10.2.2 NASA Prospecting Asteroid Mission 216
10.2.3 Other Space Swarm-Based Concepts 219
10.3 Other Applications of Swarms 219
10.4 Autonomicity in Swarm Missions 220
10.5 Software Development of Swarms 222
10.5.1 Programming techniques and tools 222
10.5.2 Verification 222
10.6 Future Swarm Concepts 224

11 Concluding Remarks 227
11.1 Factors Driving the Use of Autonomy and Autonomicity 227
11.2 Reliability of Autonomous and Autonomic Systems 228
11.3 Future missions 229
11.4 Autonomous and Autonomic Systems in Future NASA Missions 232

A Attitude and Orbit Determination and Control 237

B Operational Scenarios and Agent Interactions 241
B.1 Onboard Remote Agent Interaction Scenario 241
B.2 Space-to-Ground Dialogue Scenario 245
B.3 Ground-to-Space Dialogue Scenario 246
B.4 Spacecraft Constellation Interactions Scenario 248
B.5 Agent-based Satellite Constellation Control Scenario 252
B.6 Scenario Issues 253

Acronyms 255

Glossary 259

References 269

Index 283

Part I

Background

1

Introduction

To explore new worlds, do new science, and observe new phenomena, NASA must develop increasingly sophisticated missions. Sensitive new instruments are constantly being developed, with ever increasing capability of collecting large quantities of data. The new science performed often requires multiple coordinating spacecraft to make simultaneous observations of phenomena. The new missions often require ground systems that are correspondingly more sophisticated. Nevertheless, the pressures to keep mission costs and logistics manageable increase as well.

The new paradigms in spacecraft design that support the new science bring new kinds of mission operations concepts [165]. Ever-present competition for national resources, and the consequent greater focus on the cost of operations, has led NASA to utilize adaptive operations and move towards almost total onboard autonomy in certain mission classes [176, 195]. In NASA's new space exploration initiative, there is emphasis on both human and robotic exploration. Even when humans are involved in the exploration, human tending of space assets must be evaluated carefully during mission definition and design in terms of benefit, cost, risk, and feasibility.

Risk is a major factor supporting the use of unmanned craft: the loss of human life in two notable Shuttle disasters has delayed human exploration [160], and has led to a greater focus on the use of automation and robotic technologies where possible. For the foreseeable future, it is infeasible to use humans for certain kinds of exploration, e.g., exploring the asteroid belt, for which was posed the concept ANTS mission—discussed in Chapter 10—where uncrewed miniature spacecraft explore the asteroid belt. A manned mission for this kind of exploration would be prohibitively expensive, and would pose unacceptable risks to human explorers, due to the dangers of radiation among numerous other factors.

Additionally, there are many possible missions where humans simply cannot be utilized, for a variety of reasons, such as the long mission time line reflecting the large distances involved. The Cassini mission taking seven years to reach Titan, the most important of Saturn's moons, is an example. Another

example is Dawn, a mission to aid in determining the origins of our universe, which includes the use of an altimeter to map the surface of Ceres and Vesta, two of the oldest celestial bodies in our solar system.

More and more, these unmanned missions are being developed as autonomous systems, out of necessity. For example, almost entirely autonomous decision-making will be necessary, because the time lag for radio communications between a craft and human controllers on earth will be so large that circumstances at the craft will no longer be relevant when commands arrived back from the earth. For example, for a rover on Mars during the months of mission operations when the earth and Mars are on opposite sides of the sun in their respective orbits, there would be a round-trip delay of upwards of 40 (earth) minutes in obtaining responses and guidance from mission control.

Historically, NASA missions have been designed around a single spacecraft carrying one instrument, or possibly a small number of related instruments. The spacecraft would send its raw data back to earth to a dedicated ground control system which had a dedicated staff that controlled the spacecraft and would trouble-shoot any problems. Many of the new missions are either very complex, or have long communications delays, or require very fast responses to mission circumstances. Manual control by humans becomes problematic or impractical for such missions. Consequently, the onboard software to operate the missions is increasingly complex, placing increasing demands on software development technologies.

In anticipation of these missions, NASA has been doing research and development into new software technologies for mission operations. Two of these technologies enable autonomous and autonomic operations. Technology for autonomy allows missions to operate on their own with little or no directions from humans based on internal goals. Autonomicity builds on autonomy technology by giving the mission what is called *self-awareness*. These technologies will enable missions to go to new planets or distant space environments without constant real-time direction from human controllers.

Recent robotic missions to Mars have required constant inputs and commands from mission control to move the rovers only inches at a time, as a way to ensure that human controllers would not be too late in learning about changed conditions or circumstances or too late in returning appropriate directions to the rovers. With twenty minutes required for one-way communications between the earth and Mars, it takes a minimum of forty minutes for mission control to receive the most recent video or sensor inputs from a robot and send the next commands back. Great care is required when moving a robot on Mars or other distant location, because if it flipped over or got stuck, the mission could be ended or become severely limited. There often elapsed several hours between the receipt of new images from Mars and the transmission of movement commands back to the robot. The result of the delays was a great, but unavoidable, limitation on exploration by the robots.

If these missions could instead operate autonomously, much more exploration could be accomplished, since the rovers, reacting independently and

immediately to conditions and circumstances, would not have the long wait times for communications, or for human recalculation and discussion of the next moves. Until recently, there was not enough computational power onboard spacecraft or robots to run the software needed to implement the needed autonomy. Microprocessors with requisite radiation-hardening for use in space missions now have greatly increased computing power, which means these technologies can be added to space missions giving them much greater capabilities than were previously possible.

This book discusses the basics of spaceflight and ground system software and the enabling technologies to achieve autonomous and autonomic missions. This chapter gives examples of current and near term missions to illustrate some of the challenges that are being faced in doing new science and exploration. It also describes why autonomy in missions is needed not only from an operations standpoint but also from a cost standpoint. The chapter ends with an overview of what autonomy and autonomicity are and how they differ from each other as well as from simple automation.

1.1 Direction of New Space Missions

Many of the planned future NASA missions will use multiple spacecraft to accomplish their science and exploration goals. While enabling new science and exploration, multi-spacecraft and advanced robotic missions, along with the more powerful instruments they carry, create new challenges in the areas of communications, coordination, and ground operations. More powerful instruments will produce more data to be downlinked to mission control centers. Multi-spacecraft missions with coordinated observations will mean greater complexity in mission control. Controlling operations costs of such missions will present significant challenges, likely entailing streamlining of operations with fewer personnel required to control a spacecraft.

The following missions that have recently launched or that will be launched in the near future illustrate the types of missions NASA is planning.

New Millennium Program's Space Technology 5

The New Millennium Program's (NMP) Space Technology 5 (ST5) [171] is a technology mission that launched in March of 2006 with a 90 day mission life. The goal of the mission was to demonstrate approaches to reduce the weight, size, and therefore cost of missions while increasing their capabilities. The science it accomplished was mapping the intensity and direction of magnetic fields within the inner magnetosphere of the earth. To accomplish this, it used a cluster of three 25-kilogram class satellites (Figure 1.1). Each micro satellite had a magnetometer onboard to measure the earth's magnetosphere simultaneously from different positions around the earth. This allowed scientists to determine the effects on the earth's magnetic field due to the solar wind and other phenomena.

ST5 was designed so that the satellites would be commanded individually and would communicate directly to the ground. There was a one-week period of "lights out" operation where the the micro-sats flew "autonomously" with pre-programmed commands in a test to determine whether commanding could be done onboard instead of from ground stations. In the future, this approach would allow the spacecraft to react to conditions more quickly and save on ground control costs. Preprogrammed commands are considered only as a step toward autonomy, and as a form of automation, since the commands are not determined by internal goals and the current conditions of the spacecraft software.

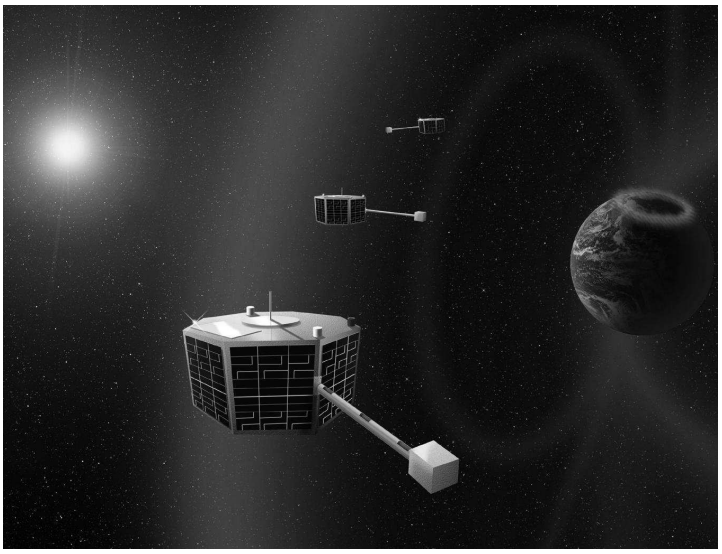

Fig. 1.1. New Millennium Program Space Technology 5 Spacecraft Mission (Image Credit: NASA).

Solar-Terrestrial Relations Observatory

The Solar-Terrestrial Relations Observatory (STEREO) mission, launched in March 2006, is studying the Sun's coronal mass ejections (CMEs), which are powerful eruptions in which as much as ten billion tons of the Sun's atmosphere can be blown into interplanetary space [37]. CMEs can cause severe magnetic storms on earth. STEREO tracks CME-driven disturbances from the Sun to earth's orbit, producing data for determining the 3D structure and dynamics of coronal and interplanetary plasmas and magnetic fields, among other things. STEREO comprises two spacecraft with identical instruments

(along with ground-based instruments) that will provide a stereo reconstruction of solar eruptions (Figure 1.2).

The STEREO instrument and spacecraft will operate autonomously for most of the mission due to the necessity of the two spacecraft maintaining exact distances from each other to obtain the needed stereographic data. The spacecraft uses a beacon mode where events of interest are identified. Broadcasts are then made to the ground for notification and any needed support. The beacon mode automatically sends an alert to earth when data values exceed a threshold level. Each of the STEREO spacecraft is able to determine its position, orientation, and orbit, and react and act autonomously to maintain proper position. Autonomous operation means significant savings over manual operation.

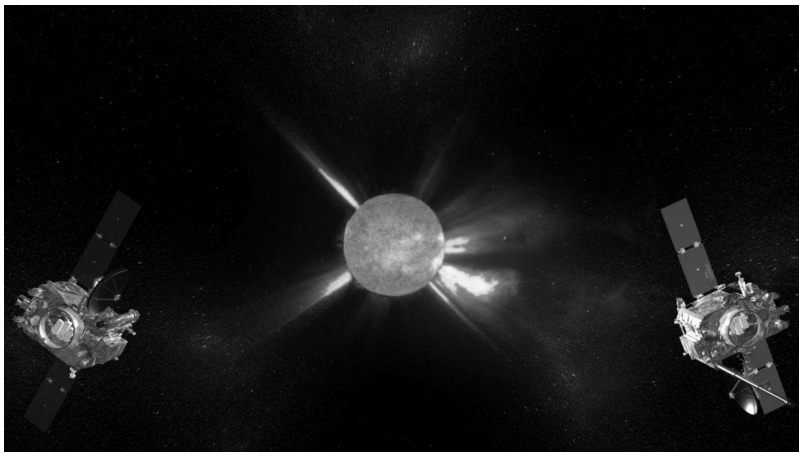

Fig. 1.2. Stereo mission spacecraft (Image Credit: NASA).

Magnetospheric Multiscale

The Magnetospheric Multiscale (MMS) mission, scheduled to launch in 2014, will use a four-spacecraft cluster that will study magnetic reconnection, charged particle acceleration, and turbulence in regions of the earth's magnetosphere (Figure 1.3). The four spacecraft will be positioned in a hexahedral or quad tetrahedral configuration with separations of 10 kilometers up to a few 10s of thousands of kilometers [51]. This arrangement will allow three-dimensional structures to be described in both the magnetosphere and solar wind.

Distances between the Cluster spacecraft will be adjusted during the mission to study different regions and plasma structures. Simultaneous measurements from the different spacecraft will be combined to produce a three-

dimensional picture of plasma structures. The spacecraft will use interspacecraft ranging and communication and autonomous operations to maintain the correct configuration and the proper distances between spacecraft.

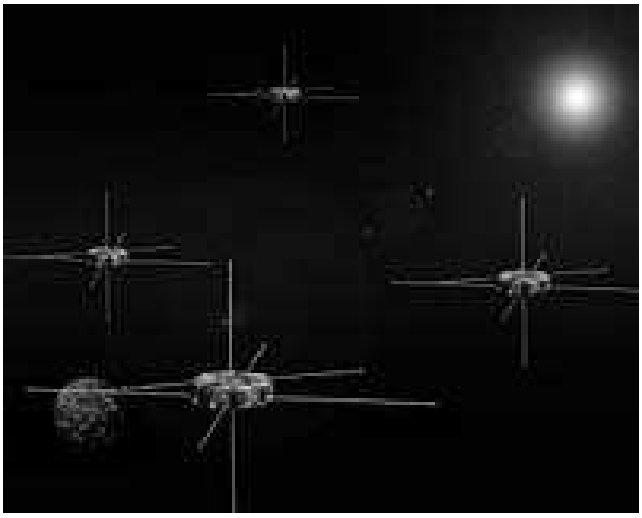

Fig. 1.3. Magnetospheric Multiscale Spacecraft (Image Credit: NASA).

Tracking and Data Relay Satellites

The Tracking and Data Relay Satellites (TDRS) relay communications between satellites and the ground (Figure 1.4) [186]. The original TDRS required ground commands for movement of the large single access antennas. TDRS-H, I and J autonomously control the antenna motion and adjust for spacecraft attitude according to a profile transmitted from the ground. This autonomous control greatly reduces the costs of operations and the need for continuous staffing of ground stations.

Other Missions

Other proposed or planned near-term missions that will have onboard autonomy include the following:

- Two Wide-angle Imaging Neutral-atom Spectrometers (TWINS) mission, which will stereoscopically image the magnetosphere,
- Geospace Electrodynamic Connections (GEC) mission, which is a cluster of three satellites that will study the ionosphere-thermosphere (2013+),
- Laser Interferometer Space Antenna (LISA) mission, which will consist of three spacecraft to study gravitational waves (2019),

Fig. 1.4. TDRS Spacecraft (Image Credit: NASA).

- Constellation X (CON-X) mission, which will provide teams of x-ray telescopes working in tandem to observe distant objects (2020), and
- Magnetotail Constellation (MC) mission, which will consist of 30+ nanostatellites that will study the earth's magnetic and plasma flow fields (2011+).

Relying on spacecraft that coordinate and cooperate with each other, NASA will be able to perform new science that would be difficult or impossible to do with a single spacecraft. And recognition of the challenges of real-time control of such complex missions would lead naturally to designing them to operate autonomously, with goals set at a higher level by human operators and scientists.

1.2 Automation vs. Autonomy vs. Autonomic Systems

In this section, the differences between automation, autonomy, and autonomicity are discussed. This establishes working definitions for this book, and aids in understanding the current state of flight automation/autonomy/autonomicity.

1.2.1 Autonomy vs. Automation

Since "autonomy" and "automation" seem to have a wide range of definitions, it is appropriate to establish how those terms will be used in the context of

this book. Both terms refer to processes that may be executed independently from start to finish without any human intervention. Automated processes simply replace routine manual processes with software/hardware ones that follow a step-by-step sequence that may still include human participation. Autonomous processes, on the other hand, have the more ambitious goal of emulating human processes rather than simply replacing them.

An example of an automated ground data trending program would be one that regularly extracts from the data archive a set list of telemetry parameters (defined by the flight operations team (FOT)), performs a standard statistical analysis of the data, outputs in report form the results of the analysis, and generates appropriate alerts regarding any identified anomalies. So, in contrast to an autonomous process, in this case the ground system performs no independent decision making based on realtime events, and a human participant is required to respond to the outcome of the activity.

An automated onboard process example could be an attitude determination function requiring no *a priori* attitude initialization. The steps in this process might be as follows. On acquiring stars within a star tracker, an algorithm compares the measured star locations and intensities to reference positions within a catalog and identifies the stars. Combining the reference data and measurements, an algorithm computes the orientation of the star tracker to the star field. Finally, using the known alignment of the star tracker relative to the spacecraft, the spacecraft attitude is calculated. The process is an automated one because no guidance from FOT personnel is required to select data, perform star identification, or determine attitudes. However, the process does not define when the process should begin (it computes attitudes whenever star data is available), simply outputs the result for some other application to use, and in the event of an anomaly that causes the attitude determination function to fail, takes no remedial action (it just outputs an error message). In fact this calculation is so easily automatable that the process described, up to the point of incorporating the star tracker alignment relative to the spacecraft, now can be performed within the star tracker itself, including compensations for velocity aberration.

On the other hand, the more elaborate process of autonomy is displayed by a ground software program that independently identifies when communications with a spacecraft is possible, establishes the link, decides what files to uplink, uplinks those files, accepts downlinked data from the spacecraft, validates the downlinked data, requests retransmission as necessary, instructs the freeing-up of onboard storage as appropriate, and finally archives all validated data. This would be an example of a fully autonomous ground process for uplink/downlink.

Similarly, a Flight Software (FSW) program that monitors all key spacecraft health and safety (H&S) data, identifies when departures from acceptable H&S performance has occurred, and independently takes any action necessary to maintain vehicle H&S including (as necessary) commanding the spacecraft to enter a safemode that can be maintained indefinitely without ground in-

tervention would be a fully autonomous flight fault detection and safemode processes.

1.2.2 Autonomicity vs. Autonomy

In terms of computer-based systems-design paradigms, autonomy implies 'self-governance' and 'self-direction', whereas autonomic implies 'self-management'. That is, autonomy (self-governance) is the delegation of responsibility to the system to meet the defined goals of the system (automation of responsibility including some decision making for the success of tasks), whereas autonomicity is system self-management (automation of responsibility including some decision making for the successful operation of the system), and as such will often be in addition to self-governance to meets one's own required goals. It may be argued at the systems level that the success of autonomy requires successful autonomicity: ultimately, ensuring success in terms of the tasks requires that the system be reliable [158].

For instance, the goals of a system may be to find a particular phenomenon using its onboard science instrument. The system may have autonomy (the self-governance/self-direction) to decide between certain parameters. The goals to ensure the system is fault tolerant and continues to operate under fault conditions, for instance, would not fall directly under this specific delegated task of the system (its autonomy), yet ultimately the task may fail if the system cannot cope with uncertain dynamic changes in the environment. From this perspective, the autonomic and self-management initiatives may be considered specialized forms of autonomy, that is, the autonomy (self-governance, self-direction) is specifically to manage the system (to heal, protect, configure, optimize, and so on).

Taking the dictionary definitions (Table 1.1) of autonomous and autonomic, autonomous essentially means 'self-governing' (or 'self-regulating', 'self-directed') as defined [76, 162]. 'Autonomic' is derived from the noun 'autonomy', and one definition of autonomous is autonomic, yet the main difference in terms of the dictionary definitions would be in terms of speed; autonomic being classed as 'involuntarily', 'reflex' and 'spontaneous'.

'Autonomic' became mainstream within computing in 2001 when IBM launched their perspective on the state of information technology [63]. Autonomic computing has four key self-managing properties[69]:

- Self-configuring,
- Self-healing,
- Self-optimizing, and
- Self-protecting.

With these four properties are four enabling properties:

Self-aware: of internal capabilities and state of the managed component,
Self-situated: environment and context awareness,

au·to·nom·ic (àwtə nómmik)
adj.

1. *Physiology.*
   a) Of, relating to, or controlled by the autonomic nervous system.
   b) Occurring involuntarily; automatic: *an autonomic reflex.*
2. Resulting from internal stimuli; spontaneous.

au·ton·o·mic·i·ty (àwtə nóm i síttee)
n.

1. The state of being autonomic.

au·ton·o·mous (aw tónnəməs)
adj.

1. Not controlled by others or by outside forces; independent: *an autonomous judiciary; an autonomous division of a corporate conglomerate.*
2. Independent in mind or judgment; self-directed.
3.
   a) Independent of the laws of another state or government; self-governing.
   b) Of or relating to a self-governing entity: *an autonomous legislature.*
   c) Self-governing with respect to local or internal affairs: *an autonomous region of a country.*
4. Autonomic.

[From Greek autonomos : auto-, *auto-* + nomos, *law*]

Table 1.1. Dictionary definitions of autonomic, autonomicity and autonomous[76].

Self-monitor and self-adjust: through sensors, effectors and control loops.

In the few years since autonomic computing became mainstream, the "self-x" list has grown as research expands, bringing about the general term "selfware" or "self-*", yet the four initial self-managing properties along with the four enabling properties cover the general goal of self management [156].

The tiers for Intelligent Machine Design [102, 136, 139, 140] consist of a top level (reflection), a middle level (routine) and a bottom level (reaction). Reaction is the lowest level, where no learning occurs, but is the immediate response to state information coming from sensory systems. Routine is the middle level, where largely routine evaluation and planning behaviors take place. It receives input from sensors as well as from the reaction level and the reflection level. Reflection, a meta-process where the mind deliberates about itself, at the top level receives no sensory input and has no motor output; it receives input from below. The levels relate to speed. For instance, reaction should be immediate whereas reflection is consideration over time. Other variations of this three-tier architecture have been derived in other research domains (see Table 1.2) [158]. Each approach is briefly discussed below.

Intelligent Machine Design	**Future Comms. Paradigm**	**DARPA/ISO's Autonomic Information Assurance**	**NASA's Science Mission**	**Self-Directing & Self-Managing System Potential**
Reflection	Knowledge Plane	Mission Plane	Science	Autonomous
Routine	Management Control Plane	Cyber Plane	Mission	Selfware
Reaction	Data Plane	Hardware Plane	Command Sequence	Autonomic

Table 1.2. Various similar 3-Tier approaches in different domains

In the future communications-paradigms research domain, a new construct, a knowledge plane, has been identified as necessary to act as a pervasive system element within the network to build and maintain high-level models of the network. These indicate what the network is supposed to do to provide communication services and advice to other elements in the network [28]. It will also work in coordination with the management plane and data planes. This addition of a knowledge plane to the existing data and management/control planes would form a three tier architecture with data, management/control, and knowledge planes [28].

In the late 1990s, DARPA/ISO's Autonomic Information Assurance (AIA) program studied defense mechanisms for information systems against malicious adversaries. The program developed an architecture consisting of three planes: mission, cyber, and hardware. One finding from the research was that fast responses are necessary to counter advance cyber-adversaries [88], similar to a reflex action discussed earlier.

As will be defined later in this book, NASA's science mission management, from a high level perspective, may be classified into:

Science Planning: setting the science priorities and goals for the mission,
Mission Planning: involving the conversion of science objectives to instrument operations and craft maneuvering, housekeeping and scheduling, communications link management, etc., and
Sequence Planning: production of detailed command sequence plans.

These versions of a high-level, three-tier view of self-governing and self-managing systems may be generalized into autonomous-selfware-autonomic tiers. Of course, this is not intended to be prescriptive nor conflict with other views of autonomic systems. The intention in examining and viewing systems in this light is to assist in developing effective systems.

1.3 Using Autonomy to Reduce the Cost of Missions

Spacecraft operations costs have increasingly concerned NASA and others, and have motivated a serious look into reducing manual control by automating as many spacecraft functions as possible. Under current designs and methods for mission operations, spacecraft send their data (engineering and science) to earth for processing and receive commands from analysts at the control center. As the complexity and number of spacecraft increase, it takes a proportionately larger number of personnel to control the spacecraft. Table 1.3 shows some current and future missions with the number of people it takes to operate them [122]. People-to-spacecraft ratios are shown (a) for past and present missions based on current technology and (b) for expected future multi-spacecraft missions with the current technology and operations approaches. The figure also shows the operator-to-spacecraft goal for the future missions. Missions capable of performing the desired science will achieve the operator-to-spacecraft ratio goals only if designed to operate without intensive control and direct commanding by human operators. Clearly, a combination of automation, autonomy, and autonomicity will be needed.[1]

In many cases, multi-spacecraft missions would be impossible to operate without near-total autonomy. There are several ways autonomy can assist multi-spacecraft missions. The following section describes some of the ways that autonomy could be used on missions to reduce the cost of operations and perform new science.

1.3.1 Multi-Spacecraft Missions

Flying multi-spacecraft missions can have several advantages, including:

- reducing the risk that the entire mission could fail if one system or instrument fails,
- making multiple observations of an object or event at the same time from multiple locations (giving multiple perspectives or making the equivalent of a large antenna from many small ones),
- reducing the complexity of a spacecraft by reducing the number of instruments and supporting subsystems, and
- replacing or adding an instrument by adding a new spacecraft into an already existing constellation or swarm.

The Microwave Anisotropy Probe (MAP) mission, launched in 2001, was forecast to use an average of four people to operate the mission (Table 1.3). This mission consists of a single spacecraft and utilizes a low number of people for operations. The Iridium satellite network has 66 satellites, and it is

[1] To simplify, in the remainder of this book, since autonomicity builds on autonomy, we will simply refer to the combination of autonomous and autonomic systems simply as autonomy, except where explicitly noted.

estimated that at the beginning of its operations, its primary operations consisted of 200 people, approximately three people per spacecraft—a 25 percent improvement over the MAP mission. The GlobalStar satellite system consists of 48 satellites and requires approximately 100 people to operate it, approximately two people per satellite. The Iridium and GlobalStar satellites are mostly homogeneous, which makes for easier operations than heterogeneous satellites. Understandably, many of the future NASA missions are proposing multiple homogeneous spacecraft.

Year	**Mission**	**Number of Spacecraft**	**Number of People to Operate with Current Technology**	**Current People: S/C**	**Goal People: S/C**
2000	MAP	1	4	4:1	–
2000	Iridium	66	200	3:1	–
2000	GlobalStar	48	100	2:1	–
2007	NMP ST5	3	12	–	1:1
2012	MC	30–40	120-160	–	1:10

Table 1.3. Ratio Goals for People Controlling Spacecraft

The last two examples in Table 1.3 show future NASA multi-satellite missions along with the operation requirement using current technology, and the goal. Keeping to the current technology of approximately four people per spacecraft, the cost of operations will become prohibitive as the number of spacecraft increases. Missions are currently being planned and proposed that will include tens and hundreds of spacecraft. The most effective way of avoiding excessive cost of operations is by reducing the operators-to-spacecraft ratio. To do this, operators need to work at a higher level of abstraction and be able to monitor and control multiple spacecraft simultaneously.

In addition to saving costs on operations, autonomy can play a vital role in reducing the size of the communications components. This in turn reduces weight and reduces the cost of the mission not only in components, but in the amount of lift needed to put the spacecraft into orbit. Historically, the mission principal investigator (PI) would want all of the data to be transmitted back to earth for archival purposes and for rechecking calculations. Earlier instruments did not generate as much data as they do now and it was not an issue in transmitting it because the onboard resources were available. Newer instruments now produce more data and many mission are now flying more remote orbits. Both of these require higher-gain antennas and more power, which increase costs, including greater cost for launch. An alternative is to do onboard processing of science data or transmit less data by stripping out non-science data, both of which result in less data to download. This is an example of one tradeoff, i.e., in basic terms, design the mission to either (a) download all of the data or (b) download only part of it (thereby reducing

the cost of one part of the mission) and doing more science by adding more instruments.

1.3.2 Communications Delays

Autonomous onboard software is needed when communications can take more than a few minutes between the spacecraft and the ground. When communications is lengthened, mission risks increase because monitoring the spacecraft in real-time (or near real-time) by human operators on earth becomes less feasible. The mission is then flown less on a real-time, current basis, and the operator needs to stay ahead of the mission by visualizing what is happening and confirming it with returned data.

Though *science of opportunity* is not necessarily restricted by communication delays, it often has required a human in the loop. This can present a problem since the phenomenon may no longer be observable by the time initial data is transmitted back to earth, a human analyzes it, and commands are sent back to the spacecraft. Autonomy can be useful in this area because it enables immediate action to be taken to observe the phenomenon. Challenges in this area include devising the rules for determining whether the new science should interrupt other on-going science. Many factors may be involved, including the importance of the current observation, the time to react to the new science (it could be over by the time the instrument reacts), the spacecraft state, and health and safety issues. The rules would have to be embedded in an onboard expert system (or other "intelligent" software) that could be updated as new rules are learned. An example of a mission that does autonomous onboard target-of-opportunity response is the Swift mission. It has a survey instrument that finds possible new gamma ray bursters, determines whether an object has high priority, and has autonomous commanding that can slew the spacecraft so the narrow-field-of-view instrument can observe it.

For spacecraft health and safety, a large communications latency could mean that a spacecraft could be in jeopardy unknown to human operators on the earth and could be lost before any corrective commands could be received from earth. As in the case of science of opportunity, many aspects need to be taken into account, need to be embedded in onboard software, and need to be updateable as the mission progresses.

1.3.3 Interaction of Spacecraft

Spacecraft that interact and coordinate with each other, whether formation flying or performing the science itself, may also have to communicate with a human operator on earth. If the spacecraft does not have onboard autonomy, the human operator performs analysis and sends commands back to the spacecraft. For the case of formation flying, having the formation coordination done autonomously or semi-autonomously through inter-spacecraft

communications can save the lag time for downloading the appropriate data and waiting for a human operator to analyze it and return control commands. In many cases, inter-spacecraft coordination can also save spacecraft resources such as power and communication. However, the more the spacecraft interact and coordinate among themselves, the larger the onboard memory needed for the state space for keeping track of the interactions.

Whether formation flying needs autonomy depends on how accurately spacecraft separations and orientations need to be maintained, and what perturbing influences affect the formation. If the requirements are loose and the perturbations (relative to the requirements) are small, the formation can probably be ground-managed, with control only applied sporadically. If the requirements are stringent and the perturbations (relative to the requirements) are large, then control will need to be exercised with minimum delay and at high rates, necessitating autonomous formation control.

1.3.4 Adjustable and Mixed Autonomy

Complete autonomy may not be desirable or possible for some missions. In these missions, adjustable and mixed autonomy may need to be used [132]. In adjustable autonomy, the level of autonomy of the spacecraft can be varied depending on the circumstances or the desires of mission control. The autonomy can be adjusted to be either complete, partial, or no autonomy. In these cases the adjustment may be done automatically by the spacecraft depending on the situation (i.e., the spacecraft may ask for help from mission control) or may be requested by mission control either to help the spacecraft accomplish a goal or to perform an action manually. Challenges in adjustable autonomy include knowing when it needs to be adjusted, as well as how much and how to make the transition between levels of autonomy.

In mixed autonomy, autonomous agents and people work together to accomplish a goal or perform a task. Often the agents perform the low level details of the task (analogous to the formatting of a paper in a word processor) while the human performs the higher-level functions (e.g., analogous to writing the words of the paper). Challenges in this area are how to get humans working with the agents, how to divide the work up between the humans and agents, and how to impart to the humans a sense of cooperation and coordination, especially if the levels of autonomy are changing over time.

1.4 Agent Technologies

Agent technologies have found themselves in many domains with very different purposes and competencies. This section discusses some of the issues involved in the design and implementation of agents, and then it will focus on three important classes: software agents, robots, and immobots (immobile robots).

Figure 1.5 lists some of the attributes used to describe agents. From the top level viewpoint, the two most important attributes of an agent are its purpose and the domain in which it operates. All other attributes can be inferred from these two. It is from these attributes that the agent will be designed and technologies selected.

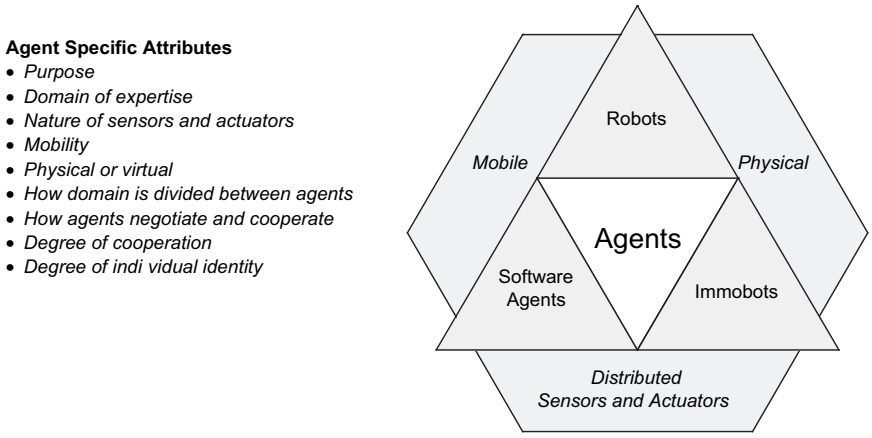

Fig. 1.5. Agent attributes.

Figure 1.5 also shows three important classes of agents. Software agents are intelligent systems that pursue goals for their human owners. An example would be an information locator that receives some objectives from its owner, interacts with electronic information sources, locates the desired information, organizes and prioritizes it, and finally presents it to the owner. Software agents exist in a virtual computer world and their sensors and actuators are distributed among the computer systems with which they interact. They may be able to migrate from one system to another, and some software agents can interact with other agents to cooperate on achieving common objectives.

Robots are mobile systems that pursue goals in the physical world. Robots are outwardly focused, with the primary goals of measuring and interacting with the external world using their sensors and actuators. Although the field of robotics has been active for many years, cooperative robotics is a relatively new area of investigation. A space-based robotics example would be the Mars Sojourner whose expertise was collecting scientific information on Mars' surface. This successful agent achieved its goals with constrained autonomy and limited ability to cooperate. After Sojourner's top level goals were downloaded from earth, it would perform multiple steps to achieve the desired objective. During task execution, it would continually monitor the situation and protect itself from unexpected events.

Immobots are immobile robotic systems that manage a distributed network of physical sensors and actuators. They are inwardly focused, with goals to monitor and maintain the general health of the overall system. There is so far only limited experience with immobots cooperating with other agents. A modern factory floor would be an example of an immobot. Integrated throughout the shop floor is a network of sensors that the immobot monitors along with actuators that are used when some action is needed. If a dangerous situation arises, the human operator must be notified and the situation explained.

These examples are all quite different but in each of them some form of computer based agency is necessary for them to carry out their mission.

The types of sensors and actuators change from agent to agent. Software Agents usually sense and manipulate the computational environment by transporting information. This information is a mixture of data to use or interpret and commands to execute. Robots' and immobots' primary sensors measure and sense the physical world and their actuators modify and interact with the physical world. In addition to their primary sensors, many robots have the ability to communicate with other agents to receive orders and report problems and results.

Many agents have mobility. Mobility in robots is easy to understand since many robots travel to achieve their objectives. Some software agents are mobile and are able to move themselves (their code and data) from one computing platform to another. Software agents may move to gain access to resources they cannot access efficiently from their original computing platform.

Agents work in either the physical or the virtual world. When in the physical world, the sensors and actuators take up space, cost money, undergo wear and tear, and consume resources while performing their mission. If the mission goal requires the physical world to be sensed and manipulated, then these costs must be paid. Software agents live in a virtual world made up of one or more computers connected by networking. In this world, moving information fulfills the roles of sensing and acting and the only direct resource consumed is computation.

1.4.1 Software Agents

Software agents are being used in many domains and encompass a wide range of technologies. At least three broad categories of software agents are being developed and applied:

Informational agents: Informational agents interact with their owner to determine the types and quantities of information the owner desires. These agents then utilize electronic sources to locate the appropriate information, which the agent then organizes and formats for presentation to the owner.

As an example, currently flight software provides subscription services so that onboard applications can subscribe to spacecraft ephemeris information and receive that information when it is calculated. An agent-based

enhancement of this arrangement might entail a scenario like the following. Suppose that predicted spacecraft ephemeris is generated onboard once per second (which is typical of current spacecraft) and represents, second by second, the best available estimation of the spacecraft position and velocity. The informational agent ensures that the information is provided once per second to applications that need it.

But occasionally the spacecraft needs to plan and schedule an orbit stationkeeping maneuver, which would be planned so as to minimize disruption to high priority onboard activities, but could (depending on orbit geometric constraints) disrupt scheduled routine onboard activities, including science. To devise the plan and incorporate the maneuver into the schedule, it needs the best predicted spacecraft ephemeris data available for an interval covering a few days in the future from the current time.

When the spacecraft's onboard maneuver agent determines that this planning must be done, the maneuver agent would contact the informational agent and describe its needs, including both the time interval to be covered and the required modeling accuracy. The informational agent then passes the request to the onboard ephemeris agent, which determines whether or not it can create a special ephemeris product that can meet the maneuver agent's needs. If the ephemeris agent can do so, the informational agent supplies it to the maneuver agent when the data becomes available.

If the accuracy requirement cannot be met using the latest extended precision seed vectors that were most recently received from the ground, the informational agent requests that the communications agent send a message to the ground station that an ephemeris update will have to be uplinked before the maneuver planning agent can do its job.

Normally, the ground would then schedule a tracking event to update the ephemeris knowledge and uplink an update to the spacecraft. However, if time is short, the ground may elect to use the improved ephemeris information to plan and schedule the next stationkeeping maneuver and uplink it (including the entire schedule with all the spacecraft attitude adjustment and subsequent rocket- or thruster-firing parameters) to the spacecraft. The informational agent would then pass on the stationkeeping maneuver (and the whole schedule itself) to the onboard scheduling agent, informing the maneuver agent that it no longer needs to worry about the next stationkeeping maneuver.

Personal assistant agents: Personal assistant agents act like a personal secretary or assistant. They know the owner's goals, schedule, and personal demands and help the owner manage his or her activities. More advanced forms of assistants can interact with other agents or humans to offload activities.

One could imagine adapting and employing a personal-assistant type of agent in the spacecraft operations domain by providing such an assistant to a planner/scheduling agent in a scenario like the following. Suppose that normally the planner/scheduler agent receives requests to schedule

various activities from other applications (attitude control, orbit maneuvering, power, communications, science instruments, etc.) and interleaves all these requests to produce a conflict-free schedule. However, occasionally a request will come either from an odd source (like a realtime ground command) or a request might arrive after the schedule has been generated and would affect the viability of activities already scheduled.

A personal assistant could be useful to the scheduling agent as a means to intercept the "out-of-the-blue" requests and figure out what to do with them. In other words, the personal assistant could decide that it is a request important enough to bother the scheduler regarding changing the existing schedule (a realtime ground command would always be that important), or could decide the request is not important enough to make a plan modification (in which case it might send it back requesting that the submitter determine whether the request could be re-submitted at a later time). The personal assistant might also determine that other agents need to be consulted and could set up a "meeting" between the various relevant agents to try to resolve the question.

Buying/negotiating agents: Buying/negotiating agents are asked to acquire some product or service. The agent interacts with potential suppliers, negotiates the best overall deal, and sometimes completes the transaction. One could imagine this type of agent adapted for use in the spacecraft operations domain, acting as an electrical power agent tasked with managing onboard power resources, in the following scenario.

Suppose that the power agent monitors overall power resources and has allocated power so as to satisfy all customers' needs (for example, attitude control subsystem (ACS), propulsion, communication, thermal, science instruments 1, 2, and 3, etc.). Suddenly the power agent realizes that a large portion of one of the solar arrays is not producing electrical power, leading to what will soon be an insufficient amount of power to satisfy all the customers' needs.

The power agent knows what the spacecraft critical functions are and immediately allocates whatever power those functions need. There remains enough power to continue to perform some science, but not all the science currently scheduled. At this point, the negotiating agent (possibly the power agent itself) "talks" with each of the three science instruments and the science scheduler to decide how best to allocate the remaining power. The scheduler points out which science (done by which instruments) are the highest priority from a mission standpoint.

The science instruments (which have already been guaranteed the power they will need to enter and maintain safemode) report what their special needs are when they transition from safemode to do their science (for example, warm-up times, re-calibrations, etc.). The scheduler then produces a draft modified schedule, which the negotiating agent checks for power validity, i.e., do they have enough power to "buy" the proposed schedule. If they do, the negotiating agent contacts ground control (through

the communications agent) to obtain ground control's blessing to change the schedule while ground control figures out how to restore (if possible) nominal power capabilities. If ground control cannot be contacted, the negotiating agent will approve executing the draft schedule until it hears from the ground or some new problem develops (or, less likely, the original problem disappears on its own).

1.4.2 Robotics

Intelligent robots are self contained mechanical systems under the guidance of computerized control systems. Intelligent robots have a long history that goes back to the beginning of computer control. While they share many features of software agents, the complex constraints placed on them by their physical environment have forced robot designers to augment the technologies used in software agents or to develop completely different architectures.

The sensors used by robots measure physical quantities such as environmental image properties, direction and speed of motion, and effector tactile feedback. They have many attributes that mean complexity in robot designs. The complex nature of the physical world makes exact sensor measurements difficult. Two readings taken moments apart or readings taken by two "identical" and "healthy" devices often differ. Physical sensors can fail in many ways. Sometime the failure will result in a device that will give correct results intermittently. Some sensors require complex processing and in many situations, the information is difficult to interpret. A vision system using even an off-the-shelf charge-coupled device (CCD) array can easily supply millions of bytes of image data every second. Image processing techniques must be used to examine the data and determine the features of the image relevant to the robot control system.

Actuators are used to make changes in the physical world. Examples are opening a valve, moving a wheel, or firing an engine. Like sensors, actuators can fail in complex ways due possibly to design deficiencies, wear and tear, or damage caused by the environment. The designers of actuators must also deal with complex interactions with the rest of the robot and the environment. For example, if a robot arm and its cargo are heavy, then actions that move the arm will also apply torque to the robot body that can significantly affect the sensor readings in other parts of the robot and can even alter the position of the robot on the supporting surface.

Since both sensors and actuators have complex failure modes, robust systems should keep long term information on the status of internal systems and develop alternative plans accounting for known failures. The unexpected will happen, so robust systems should actively detect failures, attempt to determine their nature, and plan alternative strategies to achieve mission objectives. Some systems reason on potential failures during planning in an attempt to minimize the effects that possible sensor or actuator failures could have on the ultimate outcome.

Robots exist in a world of constant motion. This requires the robot to continually sense its environment and to continually be prepared to change its plan based on unexpected events or circumstances. For example, a robotic arm attempting to pick up an object in a river bed must be prepared to adapt to changes in the object's position caused by dynamically changing river currents. Many robotic systems use reactive control systems to perform such low-level tasks. They continually sense and analyze the environment and their effect on it and, within constraints, they dynamically change their strategy until the objective is achieved. The realities of reactive control systems often make them interact poorly with the slower and symbolic, high-level control systems.

Because of their mobile nature, many robots commit a large percentage of their resources to navigation. Sensors must support detection, measurement, and analysis of all relevant aspects of the environment to enable the robot to see potential paths and recognize potential hazards. Actuators run motor and drive trains to move the robot. Finally, complex software analyzes the current situation in light of the goals and determine the best path to take to reach the destination.

1.4.3 Immobots or Immobile Robots

Immobot is a recent term described by Williams and Nayak [193]. An immobot is a large distributed network of sensors and actuators under the guidance of a computerized control system. While immobots share core robotic technologies, they differ in structure and perspective. Immobots have a robotic control system surrounded by a large number of fixed sensors and actuators connected by a network. These sensors and actuators are physically embedded in the environment they are attempting to measure and control, and the sensors can be located at great distances from the control system.

The primary objectives of immobots are different from robots. Robots spend considerable resources (hardware, consumables, and software) on externally focused activities such as navigation, sensing the world, and environmental manipulation. The immobot's sensors and actuators are fixed into their environment and their focus is internal. Their resources are dedicated to managing the environment they control to get the system into appropriate configurations to achieve objectives and to monitor and mitigate problems that occur. Immobots often monitor their systems for years, and yet when certain events occur, they need to react in real-time.

1.5 Summary

Incorporating further degrees of autonomy and autonomicity into future space missions will be crucial not only in reducing costs, but also in making some missions possible at all. The following chapters give an overview of how space

and ground software has operated in the past, the enabling technology to make autonomous and autonomic missions possible, some applications of autonomy in past systems, and future missions where this technology will be critical.

2

Overview of Flight and Ground Software

To provide a context for later chapters, this chapter presents brief summaries of the responsibilities and functionality of traditional ground and flight systems as viewed from the framework of a total system process, followed by a highlighting of the key drivers when making flight-ground trades. Details in the areas of attitude and orbit determination and control, mission design, and system engineering [47, 85, 93, 189, 191], essential for successful space missions, are beyond the scope of this book, but are well developed and interesting in their own right.

2.1 Ground System Software

Traditionally, the ground system has been responsible almost entirely for spacecraft planning and scheduling (P&S), establishment of communications for uplink and downlink, as well as science data capture, archiving, distribution, and (in some cases) processing. The ground has also shouldered the bulk of the calibration burden (both science and engineering) and much of the job of health and safety (H&S) verification. And when major onboard anomalies or failures arise, flight operation team (FOT) personnel are charged with determining the underlying cause and developing a long term solution.

So the traditional ground system has occupied the ironic position of having most of the responsibility for managing the spacecraft and its activities, and yet (with the exception of the planning and scheduling function) relying on the spacecraft to provide nearly all information required for carrying out those responsibilities. Today, in a more conventional workplace setting, this kind of work organization might be analyzed (from a reengineering perspective) to be an artificially fragmented process with unnecessary management layering leading to degraded efficiency and wasteful costs.

In the context of reengineering, the standard solution to this sort of problem is to re-integrate the fragmented process or processes by empowering the

local areas where the information originates and eliminating unneeded management layers whose only function is to shuttle that information between boxes on a classic pyramid-shaped table of organization while providing non-value added redundant checking and counter-checking.

Leaving the conventional fully earth-embedded workplace behind and returning to the modern spacecraft control center, the reengineering philosophy is still a valid one, except now the analysis of the system's fundamental processes must extend into the spacecraft's (or spacecrafts') orbit (or orbits) and must include trades between ground system functionality and flight system functionality, as will be discussed later.

A prerequisite to performing these flight-ground trades is to identify all the components of the spacecraft operations process, initially without regard to whether that component is performed onboard or on the ground. The following is a break down of operations into a set of activity categories, at least in the context of typical robotic (i.e., uncrewed) space missions. The order of the categories is, roughly, increasing in time relative to the end-to-end operations process, from defining spacecraft inputs to utilizing spacecraft outputs, although some activities (such as fault detection and correction) are continuous and in parallel with the main line.

1. Planning & Scheduling
2. Command Loading (including routine command-table uplink)
3. Science Schedule Execution
4. Science Support Activity Execution
5. Onboard Engineering Support Activities (housekeeping, infrastructure, ground interface, utility support functions, onboard calibration, etc.)
6. Downlinked Data Capture
7. Data and Performance Monitoring
8. Fault Diagnosis
9. Fault Correction
10. Downlinked Data Archiving
11. Engineering Data Analysis/Calibration
12. Science Data Processing/Calibration

Many of the operations activity groups listed above currently are partially automated at this time (for example, planning and scheduling, and data and performance monitoring), and may well become fully autonomous (within either the ground or flight systems) in the next ten years. Some of these functions are already largely performed autonomously onboard. A working definition of the difference between autonomy and automation will be supplied in Chapter 3, along with a description of the current state of the art of onboard autonomy/automation. We will next briefly discuss each of the steps in the overall spacecraft operations process.

2.1.1 Planning and Scheduling

Especially for low earth orbit (LEO) missions, the ground system planning and scheduling function traditionally has been responsible for generating a detailed desired optimized timeline of spacecraft activities, sometimes (as in the case of Hubble Space Telescope (HST)) based on rather complex predictive modeling of the spacecraft's environment and anticipated behavior. The ground system then recasts (and augments) the timeline information in an appropriate manner for processing on the spacecraft. The timeline is then executed onboard via a (largely) time-driven processor. Often along with the nominal, expected timeline, the ground interleaves it with a large array of alternate branches, to be executed in place of the nominal timeline in the event certain special conditions or anomalies are encountered. The resulting product is a highly coupled, time-dependent mass of data, which, in the past, occupied a substantial fraction of available onboard storage.

Ironically, the ground system's creation of the timeline data block is itself a process almost as highly time-dependent as the execution of the actual timeline onboard. HST provides a particularly complex example. Long-term scheduling (look-ahead intervals of several months to a year) was used to block-out accepted proposal targets within allowed geometric boundary conditions. The geometry factors are typically dominated by Sun-angle considerations, with additional contributions from issues such as moon avoidance, maximizing target orbital visibility, obtaining significant orbital dark time or low sky brightness, and meeting linkages between observations specified by the astronomer.

On the intermediate term (a few weeks to a couple of months), LEO spacecraft targets were ordered and scheduled relative to orbital events such as South Atlantic Anomaly (SAA) entrance/exit and earth occultations, and the duration of their associated observations were estimated based on required exposure time computations. Concurrently, support functions requiring scheduling, like communications, were interleaved with the science-target scheduling.

Lastly, on the short term (a day to a week), final detailed scheduling (both of science targets and support functions) to a precision of seconds was performed using the most accurate available model data (for example, the most recent predicted spacecraft ephemeris), with the possibility for including new targets (often referred to as targets of opportunity (TOOs)) not previously considered.

At times, the software needed to support the intermediate and short term scheduling process performed by the ground system has been massive, complex, and potentially very brittle. Further, multiple iterations of the process, frequently involving a lot of manual intervention (at considerable expense), was often required to produce an error-free schedule. Although considerable progress has been made in streamlining this process and reducing its associated costs, the mathematical modeling remains fairly sophisticated and some

amount of operational inefficiency is inevitable due to the necessity of relying on approximations during look-ahead modeling.

2.1.2 Command Loading

By contrast to the planning and scheduling function, command loading is quite straightforward. It consists of translating the directives output from planning and scheduling (plus any realtime directives, table loads, etc., generated at and output from the control center) into language/formats understandable by the flight computer and compatible with the communications medium. As communications protocols and the input interfaces to flight computers become more standardized, this ground system function will become steadily more automated via commercial off the shelf (COTS) tools.

2.1.3 Science Schedule Execution

Science schedule execution refers to all onboard activities that directly relate to performing the science mission. They include target acquisition, science instrument (SI) configuration, and SI operation on target (for example, exposure time management).

2.1.4 Science Support Activity Execution

Science support activities are those that are specifically performed to ensure the success of the science observation, but are not science observations themselves, nor are they routine housekeeping activities pertaining to maintenance of a viable observing platform. They are highly mission/science instrument specific activities and may include functionality such as optical telescope assembly (OTA) calibration and management and SI direction of spacecraft operation (such as pointing adjustment directives). These activities may be performed in immediate association with ongoing science, or may be performed as background tasks disjoint from a current observation. Although executed onboard, much (if not all) of the supporting calculations may be done on the ground and the results uplinked to the flight computer in the form of tables or commands.

2.1.5 Onboard Engineering Support Activities

Onboard engineering support activities are routine housekeeping activities pertaining to maintenance of a viable observing platform. The exact form of their execution will vary from spacecraft to spacecraft, but general categories are common within a mission type (e.g., Geosynchronous Earth Orbit (GEO) earth-pointer, LEO celestial-pointer, etc.). Housekeeping activities include angular momentum dumping, data storage and management, attitude and orbit

determination and/or prediction, attitude control, and orbit stationkeeping. These activities may be performed in immediate association with ongoing science, or may be performed as background tasks disjoint from a current observation. Again, although executed onboard, some of the supporting calculations may be done on the ground and the results uplinked to the flight computer in the form of tables or commands.

2.1.6 Downlinked Data Capture

Capture of downlinked telemetry data is rather straightforward and highly standardized. This ground system function will become steadily more automated via COTS tools.

2.1.7 Performance Monitoring

Monitoring of spacecraft performance and H&S by checking the values of telemetry points and derived parameters is a function that is currently shared between flight and ground systems. While critical H&S monitoring is an onboard responsibility (especially where triggers to safemode entrance are concerned), the ground, in the past, has performed more long term, non-realtime quality checking, such as hardware component trending, and accuracy analysis, as well as analysis of more general performance issues (e.g., overall observing efficiency).

2.1.8 Fault Diagnosis

Often the term "fault detection and correction (FDC)" has been used in connection with spacecraft H&S autonomy. Such terminology tends to conceal an important logical step in the process, which in the past has been exclusively the preserve of human systems engineers. This step is the diagnosis of the fundamental cause of problems based on measured "symptoms".

Traditionally, prior to launch, the systems engineers would identify a whole host of key parameters that needed to be monitored onboard, specify tolerances defining in-range vs. out-of-range performance, and identify FSW responses to be taken in realtime and/or FOT responses to be taken in near-realtime. But what has actually occurred is that the systems engineers have started with a set of failure scenarios in mind, identified the key parameters (and their tolerances/thresholds) that would measure likely symptoms of those failures, figured out how to exclude possible red-herrings (i.e., different physical situations that might masquerade as the failure scenario under consideration), and (in parallel) developed corrective responses to deal with those failures. So the process of transitioning from the symptoms specified by the parameters to the correction action (often a static table entry) that constitutes the diagnosis phase conceptually actually occurs (pre-launch) in the reverse order and the intellectual content is sketchily stored rigidly onboard.

In the post-launch phase, the systems engineers/FOT may encounter an unanticipated problem and must perform a diagnosis function using the telemetry downlinked to the ground. In such cases, operations personnel must rely on their experience (possibly with other spacecraft) and subject matter expertise to solve the problem. When quick solutions are achieved, the process often used is that of pattern recognition (or, more formally, case-based reasoning, as will be discussed later), i.e., the recognition of a repetition of clues observed previously when solving a problem in the past. The efficient implementation of a capability of this sort lies in the domain of artificial intelligence. Failing at the pattern recognition level, a more lengthy general analytical phase (the human equivalent of state modeling) typically ensues that is manually intensive and at times very expensive.

So a spacecraft called upon to diagnose and isolate its own anomalies is being asked not just to emulate the capabilities of human beings. In fact, it is being asked to emulate the capabilities of the most senior and knowledgeable individuals associated with the operations staff. Therefore, as a FSW implementation of this function must by its nature be extremely costly, a very careful trade must be conducted prior to migrating this function to the spacecraft.

2.1.9 Fault Correction

Currently, generating a plan to correct an onboard anomaly, fault, or failure is exclusively a ground responsibility. These plans may be as simple as specification of a mode change or as complex as major hardware reconfiguration or FSW code modification. In many cases, canned solutions are stored onboard for execution in response to an onboard trigger or ground command, but creation of the solution itself was done by ground system personnel, either in immediate response to the fault or (at times) many years prior to launch in anticipation of the fault. And even where the solution has been worked out and validated years in advance, a conservative operations philosophy has often kept the initiation of the solution within the ground system. So, although future technical improvements in onboard computing power and artificial intelligence tools may allow broader onboard independence in fault correction, major changes in operations management paradigms will be needed before we see more widespread migration of this functionality onboard.

2.1.10 Downlinked Data Archiving

Archiving of downlinked telemetry data (including, in some cases, distribution of data to users) is rather straightforward and highly standardized. This ground system function will become steadily more automated via COTS tools.

2.1.11 Engineering Data Analysis/Calibration

Traditionally, nearly all spacecraft engineering analysis and calibration functions (with the exception of gyro drift-bias calibration and, for the Small Explorer (SMEX) missions, magnetometer calibration) have been performed on the ground. These include attitude-sensor alignment and polynomial calibrations, battery depth-of-discharge and state-of-charge analyses, communications-margins evaluations, etc. Often the work in question has been iterative and highly manually intensive. Some progress has been made towards the further automating of at least portions of these tasks, yielding reduced production costs. It appears at this time to be less significant, from a purely cost basis, as to whether this functionality is performed onboard or on the ground

2.1.12 Science Data Processing/Calibration

Science data processing and calibration has been nearly exclusively a ground system responsibility for two reasons. First, the low computing power of radiation hardened onboard computers (OBCs) relative to that available in ground systems has limited the degree to which science data processing can be performed onboard. Second, the science community generally has insisted that all the science data be brought to the ground. Their position arises from a concern that the data might not be processed as thoroughly onboard as it might be on the ground, and that the science data users often process the same data multiple times using different algorithms, calibrations, etc., sometimes years after the data was originally collected.

Given the science customers' strong views on this subject, independent of potential future advances in radiation hardened processing capabilities, it would be ill-advised to devise a mission concept that relies exclusively on such onboard autonomy features. A more appropriate approach would be to offer these features as options to users, in effect allowing them to take advantage of cost-saving opportunities as they deem appropriate. One can envision a dual scenario where deep space (DS) missions not only would send back the raw data for the science community, but also would process it onboard to permit the exercise of onboard autonomy through which the spacecraft might spot potential TOOs and take unplanned science observations without having to wait for possibly (likely) untimely instructions from ground control.

2.2 Flight Software

Although highly specialized to serve very precise (and often mission-unique) functions, FSW must simultaneously satisfy a broad range of competing needs. First, it is the FSW that provides the ground system an interface with the flight hardware, both engineering and science. Since spacecraft hardware components (including science instruments) are constantly being upgraded as

their technologies continue to advance, the FSW elements that communicate to the hardware components must regularly be updated as well. Fortunately, as the interface with the ground system has largely been standardized, the FSW elements that communicate to the ground remain largely intact from mission to mission. It is in fact the ability of the FSW to mask changes in flight hardware input/output channels that has provided the ground system a relatively stable environment for the development of standardized COTS products, which in turn has enabled dramatic reductions in ground system development costs.

Second, it is the responsibility of the FSW to respond to events that the ground system cannot deal with, because

1. the spacecraft is out of contact with the ground,
2. the response must be immediate,
3. critical spacecraft or payload issues are involved, or
4. the ground lacks key onboard information for formulating the best response.

Historically, the kinds of functions allocated to FSW for these reasons were ones such as the attitude control subsystem (ACS), safemode processing and transition logic, fault detection and correction, target acquisition logic, etc.

Third, the FSW can be used to streamline (at least in part) those (previously considered ground system) processes where an onboard, autonomous response is cheaper or more efficient. In many of these cases, routine processes may first be performed manually by operations personnel, following which automated ground software is developed to reduce costs. After the automated ground process has been fully tested operationally, the software or algorithms may then be migrated to the flight system where further cost reductions may be achievable.

Fourth, the process may be performed onboard in order to reduce demand on a limited resource. For example, downlink bandwidth is a valuable, limited quantity on most missions, either because of size/power constraints on spacecraft antennas/transmitters, or because of budget limitations on the size of the ground antenna. In such cases, FSW may be used to compress the output from payload instruments or prune excessive detail from the engineering telemetry stream to accommodate a smaller downlink data volume.

As can be seen from even casual consideration of these few examples, the demands placed on FSW have a widely varying nature. Some require high precision calculation of complex mathematical algorithms. These calculations often must be performed extremely quickly and the absolute time of the calculation accurately placed relative to the availability of key input data (here, we are referring to the data-latency issue). On the other hand, some FSW functions must process large quantities of data or must store and manage the data. Other functions must deal with intricate logic trees and orchestrate realtime responses to anomalies detected by self-monitoring functions. And because the FSW is the key line of defense protecting spacecraft H&S, all

these functions must be performed flawlessly and continuously, and for some missions (due to onboard processor limitations) must be tightly coupled in several processing loops.

The following is a list of the traditional FSW functions:

1. Attitude determination and control
2. Sensor calibration
3. Orbit determination/navigation (traditionally orbit maneuver planning has been a ground function)
4. Propulsion
5. Executive and task management
6. Time management
7. Command processing (target scheduling is traditionally a ground function)
8. Engineering and science data storage and handling
9. Communications
10. Electrical power management
11. Thermal management
12. Science instrument commanding
13. Science instrument data processing
14. Data monitoring (traditionally no trending)
15. Fault detection and correction (FDC)
16. Safemode (separate ones for spacecraft and payload instruments)

2.2.1 Attitude determination and control, Sensor Calibration, Orbit Determination, Propulsion

Often in the past, attitude determination and control, sensor calibration, orbit determination/navitation, and propulsion functions have resided within a separate attitude control subsystem (ACS) processor because of the high central processing unit (CPU) demands of its elaborate mathematical computations. As onboard computer (OBC) processing power has increased, this higher cost architecture has become more rare, and nowadays a single processor usually hosts all the spacecraft bus functions. Attitude determination includes the control laws responsible for keeping the spacecraft pointing in the desired direction and reorienting the spacecraft to a new direction. Currently, onboard attitude sensor calibration is limited to gyro drift-bias calibration (and for some spacecraft a coarse magnetometer calibration).

Orbit determination may be accomplished by measurement (Global Positioning System (GPS) for example), solving the equations of motion, or by use of an orbit propagator. Traditionally, orbit maneuver planning has been the responsibility of the ground, but some experiments have been performed migrating routine stationkeeping-maneuver planning onboard, e.g., Earth Observing - 1 (EO-1). Regardless of whether the orbit-maneuver planning is done

onboard or on the ground, the onboard propulsion subsystem has responsibility for executing the maneuvers via commands to the spacecraft's thrusters, which also at times may be used for attitude control and momentum management.

2.2.2 Executive and Task Management, Time Management, Command Processing, Engineering and Science Data Storage and Handling, Communications

Command and data handling (C&DH) includes the executive, time management, command processing, engineering- and science-data storage, and communication functions. The executive is responsible for coordinating and sequencing all the onboard processing, and separate local executives may be required to control lower level processing within a subsystem. The command processor manages externally supplied stored or realtime commands, as well as internally originated commands to spacecraft sensors, actuators, etc. Again, depending on the design, some command management may be under local control.

The C&DH also has management responsibility for engineering- and science-data storage, in the past via tape recorders but nowadays via solid state storage. Depending on the level of onboard sophistication, much of the bookkeeping job may be shared with the ground, although the trends are towards progressively higher levels of onboard autonomy. Telemetry uplink and downlink are C&DH responsibilities as well, although articulation of moveable actuators (such as high gain antenna (HGA) gimbals) as well as any supporting mathematical modeling associated with communications (e.g., orbit prediction) are typically the province of the ACS.

2.2.3 Electrical Power Management, Thermal Management SI Commanding, SI Data Processing

Critical H&S functions like spacecraft electrical power and thermal management are usually treated as separate subsystems, although the associated processing may be distributed among several physical processor locations (or located in the spacecraft bus processor) depending on the design of the flight system. This distribution of sub-functionality is particularly varied with regards to SI commanding and data processing, given the steadily increasing power of the SIs' associated microprocessors. Currently, any onboard processing that is associated with a spacecraft's Optical Telescope Assembly (OTA) falls within the context of the SI functions, although as OTA processing becomes more autonomous with the passage of time, it could well warrant independent treatment.

2.2.4 Data Monitoring, Fault Detection and Correction

The processing associated with data monitoring and FDC are even more highly distributed. Typically, the checking of individual data points and the identification of individual errors (with associated flag generation) is done locally, often immediately after the measurement is read out from its sensor. On the other hand, fault correction is typically centralized so responses to multiple faults can be dealt with in a systematic manner.

2.2.5 Safemode

The last item, safemode, may include several independent sub-functions, depending on the cost and complexity of the spacecraft in question. Typical kinds of safemode algorithms include Sun acquisition modes (to maintain power positive, maintain healthy thermal geometry, and protect sensitive optics), spin-stabilized modes (to maintain attitude stability), and inertial hold mode (to provide minimal perturbation to current spacecraft state). Usually, the processing for one or more of these modes is located in the main spacecraft bus processor, but often in the past there has been a fall back mode in a special safemode processor, the attitude control electronics (ACE) processor, in case the main processor itself has gone down. In addition to its safemode responsibilities, the ACE was the interface with the attitude sensors and actuator hardware, obtaining their output data, and providing command access. The individual SIs themselves also have separate safemode capabilities, executed out of their own processor(s). Anomalies causing the main bus processor to become unavailable are dealt with via a special uplink-downlink card that in the absence of the main processor enables continued (though limited) ground communication with the spacecraft.

2.3 Flight vs. Ground Implementation

Increasing levels of onboard autonomy are being enabled by increases in flight data system capacities (CPU, Input/Output (I/O), storage, etc.), as well as by the new approaches/structures for FSW design and development (object-oriented design, expert systems, remote agents, etc.). In particular, operational activity categories that previously were virtually the private domain of the ground systems (such as planning and scheduling, engineering data analysis and calibration, and science data processing and calibration) now provide exciting opportunities for shifting responsibility from the ground to the flight component in order to take advantage of the strengths inherent in a realtime software system in direct contact with the flight hardware.

The key advantages possessed by the flight component over the ground component are immediacy, currency, and completeness. Only the flight component can instantly access flight hardware measurements, process the infor-

mation, and respond in realtime. For example, for performance of basic spacecraft functions such as attitude control and thermal/power management, only the FSW has direct access in realtime to critical information needed to define the spacecraft's operational state, as well as the direct access to the spacecraft actuator hardware required to create and maintain the desired state. The FSW is also the only component of the integrated flight/ground operational system with full-time access to all relevant information for applications such as fault detection and SI target acquisition.

By contrast, in the past, the advantage of the ground over the flight segment has been the larger, more powerful ground computers that (for example) have enabled the ground system to execute extremely intricate schedule optimization algorithms using highly complex predictive models. However, as the power of the flight computers continues to grow with time, a partial shift of even some of these traditional ground monopolies may be justified to take advantage of the realtime information exclusively available onboard. In fact, as these hardware differences between the two platform environments narrow, the distinction between flight-based vs. ground-based may begin to blur somewhat, bringing with it the potential for more mission-customized allocation of functions between flight and ground systems.

3

Flight Autonomy Evolution

As new ideas surface for implementing advanced autonomous functions onboard spacecraft, the extent to which spacecraft already possess autonomous capability is often not fully appreciated. Many of these capabilities, in fact, have been in place for so long they have become absorbed within the flight software (FSW) infrastructure, and as a result typically are not even considered when FSW autonomy is discussed.

Another aspect of flight autonomy not often formally recognized is that the current state of flight autonomy is actually the product of an implicit process driven by the users and developers of FSW. Each autonomous function in place onboard NASA spacecraft has been developed either in response to the needs of the users of spacecraft, both the science users and the flight operations team (FOT), or in response to FSW development team insights into how their product can be made more useful to its customers. Because of the rightfully conservative nature of all three groups (scientists, FOT, and FSW developers), the pace of autonomy introduction tends to be measured, evolutionary, and targeted to very specific needs and objectives, rather than sweeping and revolutionary.

Also, the budget process, which typically targets funds to the performance of individual missions rather than allocating large research and development (R&D) funds for the development of generic functionality for future missions, tends to select against funding of major change and selects for funding of incremental change. As mission budgets have steadily shrunk, funds available to mission project managers must be dedicated more to flight-proven autonomy functionality applicable to meeting immediate mission needs, as opposed to being used for risky, breakthrough autonomy concepts that might greatly reduce costs of both the current and other missions.

To provide a somewhat more balanced perspective on this issue, the evolving role of flight autonomy in spacecraft operations will be described within the context of uncrewed space missions from the following perspectives:

1. The reasons for providing flight autonomy and what autonomous capabilities have been developed to support these needs
2. The general time frame in which these capabilities have been developed
3. Possible future trends in flight autonomy development

While the material in this chapter relates particularly to science missions (e.g., astronomical observatories, communications satellites, and satellites that observe the surface of the earth), it should be applicable as well to the development of missions that conduct robotic explorations of planets, moons, asteroids, etc. Many of the philosophies, cultures, budgetary constraints, and technologies span across all groups and agencies developing and flying any type of uncrewed mission. Brought to the fore by *crewed* missions, however, are numerous different considerations and constraints under which it makes little sense to try to design crewed assets with autonomy. Consequently, crewed missions are beyond the scope of this book.

3.1 Reasons for Flight Autonomy

Flight autonomy capabilities typically are developed at NASA Goddard Space Flight Center (GSFC) in order to

1. Satisfy mission objectives
2. Support spacecraft operations
3. Enable and facilitate efficient spacecraft operations

The object of a science mission, of course, is to perform the science for which the spacecraft has been constructed in an accurate and efficient manner. As the lifetime of a spacecraft (and the mission as a whole) is limited both by onboard hardware robustness and budget allocations, it is crucially important to optimize science data gathering as a function of time. And since all science observations are not of equal importance, the optimization strategy cannot be simply a matter of maximizing data bit capture per unit time. So a GSFC science mission must be conducted in such a manner as to support efficient, assembly line-like collection of data from routine, pre-planned observations while still permitting the flexibility to break away from a programmed plan to exploit unforeseen (in the short term) opportunities to perform time-critical measurements of ground-breaking significance.

Reconciling these inherently contradictory goals requires a subtle interplay between flight and ground systems, with carefully traded allocations of responsibility in the areas of schedule planning and execution. Typically, the superior computing power of the ground system is utilized to optimize long- and medium-range planning solutions, while the quick responsiveness of the flight system is used to analyze and react to realtime issues. In the next section, we will discuss what autonomous capabilities have been developed to support the flight system's role, and how they enable higher levels of efficiency and flexibility.

To achieve all of its mission objectives, a spacecraft must maintain near-nominal performance over a minimum lifetime. This day-to-day maintenance effort implies the presence of an onboard infrastructure supporting routine activities, such as command processing and resource management, as well as performance of these activities themselves. Furthermore, since a spacecraft's lifetime may be compromised by onboard hardware failures, latent software bugs, and errors introduced operationally by the ground system, the infrastructure must include the capability to safeguard the spacecraft against further damage or loss from these causes. Therefore, the flight system must routinely monitor spacecraft health and, on detection of an anomaly, either fix the problem immediately so that the mission can continue, or configure the spacecraft so that (at a minimum) it remains in a benign, recoverable state until the problem can be analyzed and solved by operations support personnel.

In addition, to accomplish its mission objectives as economically as possible, the entire system (spacecraft platform and payload, flight data system, and ground system) must be developed not only within increasingly stringent cost constraints, but must also be designed so as to make the conduct of the mission as inexpensive as possible over the entire mission lifetime. To achieve this, the flight system must be designed to carry out spacecraft activities accurately, efficiently, and safely, while at the same time performing these activities in a manner that reduces the complexity and mission-uniqueness of the flight and ground system, and facilitates the reduction of FOT staffing during routine operations.

In the following sections, each of these three major drivers for flight autonomy will be examined relative to more specific objectives or goals to be achieved, and the means by which these goals are achieved. A summary of this breakdown is provided[1] in Table 3.1.

3.1.1 Satisfying Mission Objectives

GSFC spacecraft mission objectives can be grouped in three major classifications:

1. Science execution,
2. Resource management, and
3. Health and Safety maintenance.

Put briefly, these objectives encompass what must be done to perform science efficiently, what onboard resources must be managed in support of science execution, and what precautions must be taken to safeguard the spacecraft while these activities are being performed. The flight system has a critical

[1] In this table, and frequently in this book, the simple term "ground" will be used to mean "ground system" or "ground operations" in reference to personnel and spacecraft control capabilities on earth.

Table 3.1. Reasons for Flight Autonomy

Reason	**Objective**	**Means**
Satisfy Mission Objectives	Efficient Science Execution	stored commanding
		autonomous pointing control
		autonomous target acquisition
		onboard science data packaging
	Efficient Resource Management	manage computing power
		manage internal data transfer
		manage time
		manage electrical power
		manage data storage
		manage telemetry bandwidth
		manage angular momentum
		manage propulsion fuel
	Health & Safety Maintenance	monitor spacecraft functions
		identify problems
		institute corrections
Support Infrastructure	Command Validation	verify transmission accuracy
		associate command with appl.
		verify arrival at application
		check content validity
		verify successful execution
	Request Orchestration	stored commands
		realtime ground requests
		autonomous onboard commands
		event-driven commanding
	Efficient Resource Management	see list above
	H&S Maintenance	see list above
Efficient Spacecraft Operations	Access to S/C Systems	commanding infrastructure
		model parameter modification
		FSW code modification
	Insight into S/C Systems	optimized telemetry format
		multiple telemetry formats
		telemetry filter tables
	Lifecycle Cost Minimization	remove ground from loop
		break couplings between functions
		exploit s/c realtime knowledge

role to play in each of these areas, both by reliably and predictably executing the ground system's orders and by autonomously reacting to realtime events to provide enhanced value above and beyond a "simple-minded" response to ground requests. In the following subsections, each of the three objectives will be discussed in a general fashion, with some specific examples cited to illustrate general concepts and principles. Later, these topics will be touched upon again in a more detailed fashion when current and possible future flight system autonomous capabilities are discussed.

Flight Autonomy Enablers of Efficient Science Execution

For science that is predictable and can be scheduled in advance (e.g., science that is characteristic of a LEO celestial-pointer spacecraft), the objective is to pack as much science into the schedule as possible with minimal time overhead or wastage. Traditionally, it is the responsibility of the ground system to solve the "traveling salesman" problem by generating an error-free schedule that optimizes data collection per unit time elapsed. Although schedule generation is the most complex part of the problem, the ground cannot cause its schedule to be executed effectively without the cooperation and support of several autonomous flight capabilities.

Command Execution

First, the flight system must execute the activities specified by the ground at the required times. Traditionally, the ground-generated schedule has defined both the activities and their execution times in a highly detailed manner. An activity can be viewed as a collection of directives that specify the steps that must be performed for the activity to be completed successfully. The directives themselves are decomposed into actual flight hardware or software commands that cause the directives to be executed. Depending on the sophistication of the flight system, the decomposition of the directives into commands may be done by the ground system and uplinked in detail to the spacecraft, or may be specified at a much higher level by the ground system, leaving the decomposition job to the flight system. In practice, most missions share the decomposition responsibility between ground and flight systems, trading onboard storage resources vs. onboard processing complexity.

However the decomposition issues are decided, the collection of directives and commands are uplinked and stored onboard until executed by the flight system in one of the following three ways: absolute-timed, relative-timed, or conditional. Note that this discussion is limited to stored commanding, as opposed to other approaches for commanding the spacecraft in realtime. The way the FSW orchestrates the potentially competing demands of stored commanding with realtime commanding—the commanding originating externally from the ground as well as internally from within the flight system—will be touched upon in later sections dealing with FSW infrastructure.

Absolute-timed commands include an attached time specified by the ground defining precisely when that command is to be executed. This approach is most appropriate for a ground scheduling program that accurately models both planned spacecraft activities as well as ground track and external events or phenomena. It allows an extremely efficient packing of activities, provided no serious impacts occur due to unforeseen events. Major anomalous events would break the plan and potentially invalidate all downstream events. In such cases, science observations (especially for LEO celestial-pointers) might have to be postponed until the ground scheduling system can re-plan and intercept the timeline. Alternatively, for events only affecting the current observation, the spacecraft could simply skip the impacted observation and recommence science activities (supported by any required engineering activities, such as dish-antenna slewing) at the start time of the next observation. This latter alternative can be particularly appropriate for earth-pointers, where the spacecraft's orbit will automatically carry it over the next target. For such cases, resolving a problem in the scheduled timeline can simply involve "writing off" the problem target and reconfiguring the spacecraft and SI so that they are in the correct state when the orbit ground track carries them over the next target. Minor anomalous events can be handled by padding the schedule with worst-case time estimates, thereby reducing the operational efficiency, or uplinking potentially extensive "what if" contingency scenarios, increasing demands on onboard storage.

Use of relative-timed commands reduces somewhat the accuracy demands on ground-system modeling. The ground system still specifies an accurate delta-time (with respect to a spacecraft or orbital event) for executing the activity, but the flight system determines when that key event has occurred. Although the treatment of timing issues differs in the two cases (absolute- vs. relative-timed), the treatment of the activity definition (i.e., how the activity is decomposed into directives and commands) would remain the same.

By contrast, conditional commanding requires the flight system to make realtime decisions regarding which directives or commands will be executed, as well as *when* they will be executed. When conditional commanding is employed, the ground specifies a logic tree for execution of a series of directives and/or commands associated with the activity, but the flight system determines when the conditions have been met for their execution, or chooses between possible branches based on observed realtime conditions. For current GSFC spacecraft, conditional commanding typically is used for detailed-level commanding within a larger commanding entity (e.g., an activity), with time-padding used to ensure that no time conflicts will occur, regardless of what decisions are made by the flight system within the conditional block. These various commanding methods provide an infrastructure enabling accurate and effective execution of ground-specified activities. Traditional applications utilizing this commanding infrastructure include pointing control and SI configuration. Conditional commanding can also enable more flexible onboard planning and scheduling functions than would be achieved through absolute-

timed commanding, permitting selective target scheduling and autonomous target rescheduling (e.g., High Energy Astronomical Observatory-2's (HEAO-2's) very flexible onboard scheduling scheme driven by its target list).

Pointing Control

The second major enabler of science execution is autonomous pointing control. Without autonomous spacecraft pointing control, efficient spacecraft operations would be impossible. In fact, the survival of the spacecraft itself would be highly unlikely for other than passively stabilized spacecraft. For spacecraft requiring active attitude control, it is the exclusive responsibility of the flight system to maintain the spacecraft at the desired fixed pointing within accuracy requirements, to reorient the attitude to a new pointing (as specified by the ground), or (in the case of survey spacecraft) to cause the attitude to follow a desired trajectory. Put somewhat more simply, it is the flight system's job to point the spacecraft in the direction of a science target and maintain that pointing (or pointing trajectory) throughout the course of the science data collection.

Once the required spacecraft orientation for science activities has been achieved, the SI(s) must be configured to support target identification, acquisition, and observation. Although target identification and acquisition can be performed with the ground system "in the loop" for missions where realtime communications are readily available and stable, typically optimization of operations requires that these functions be conducted autonomously by the flight system. For most missions, routinely supplying realtime science data to the ground is precluded by orbit, geometry considerations, timing restrictions, and/or cost.

As a general rule, the flight system will autonomously identify the science target (sometimes facilitated by small attitude adjustments) and acquire the target in the desired location within the field of view (FOV) of the SI (also, at times, supported by small spacecraft re-orientations). Once these goals have been achieved, the flight system will configure the SI(s) to perform the desired observation in accordance with the activity definition uplinked by the ground. Note that the flight system may even be assigned some of the responsibility for defining the details for how the science observation activity should be performed. This autonomy is enabled by relative-timed and conditional commanding structures. For example, for a LEO spacecraft whose SIs are adversely impacted by energetic particles within the South Atlantic Anomaly (SAA), the flight system could determine start and stop times for data taking relative to exit from and entrance into SAA contours. Or, based on its realtime measurement of target intensity, the flight system could determine via conditional commanding how long the SI needs to observe the target to collect the required number of photons.

Data Storage

Once a target has been successfully acquired and science data starts flowing out of a SI, the flight system must store the data onboard in a manner such that unnecessary burdens are not forced on the ground system's archive and science data processing functions. To that end, the FSW may evaluate science data as it is generated. Data passing validation can then be routed to onboard storage, while data failing validation can be deleted, saving onboard storage space, downlink bandwidth, and ground system processing effort. In practice, a requirement of many science missions is that all the raw science data be downlinked, so often this potentially available flight capability is not implemented or exercised. Even for those missions, however, lossless compression of data may be performed onboard (usually in hardware), yielding significant savings in onboard space and bandwidth (as much as a 3-to-1 reduction in data without loss of information content), while at the same time affording the science customer full information, even to the point of backing out the original raw science data. Finally, the flight system can play a valuable role by exploiting the realtime availability (onboard) of both science and engineering data to synchronize time-tagging and even to package data into organized files tailored to the needs of the customer for whom the data is targeted.

Flight Autonomy Enablers of Efficient Resource Management

In addition to these fundamental applications (command execution, pointing control and data storage) that are the primary components of conducting science, the flight system must also support auxiliary applications associated with managing limited onboard resources, including computing power, internal data transfer, electrical power, data storage, telemetry downlink bandwidth, angular momentum, and rocket/thruster propellant.

Computing Power and Internal Data Transfer

The first two items, computing power and internal data transfer, are managed both through the design of the FSW and by realtime monitoring of FSW performance. Traditionally, at its high level design, FSW functions have been carefully scheduled so as to ensure adequate computational resources are available to permit completion of each calculation within timing requirements without impacting the performance of other calculations. Although the FSW may often be operated below peak intensity levels, the FSW is designed to be capable of handling worst case demands. Similarly with respect to internal data flows, the bus capacities are accounted for when analyzing the feasibility of moving calculation products, sensor output, and commands through the flight data system. To deal with anomalous or unexpected conditions causing "collisions" between functions, from either a CPU or I/O standpoint, the flight system monitors itself in realtime and, in the event of a conflict,

will autonomously assign priority to those functions considered most critical. Should an acceptable rationing of resources not prove possible, or if the conflict persists for an unacceptably long period of time, the flight system (or a major component of the flight system, such as an individual SI) will autonomously transition itself to a state or mode of reduced functionality (usually a safemode) with correspondingly lower CPU and/or I/O demands.

Power Management

The flight system plays a role in electrical power management at three levels: positioning solar arrays (SAs), managing battery charging and discharging, and overall power monitoring and response. For celestial-pointing spacecraft having movable SAs, the FSW will select the appropriate SA position for each attitude that produces the desired energy collection behavior. Usually the position is chosen to optimize power generation, although for missions where over-charging batteries is a concern, the FSW may offset the SA(s) from their optimal position(s). For earth-pointing spacecraft, the FSW will rotate the SA to track the Sun as the spacecraft body is rotated oppositely so as to maintain nadir pointing. The FSW will also autonomously control battery discharging and charging behavior consistent with algorithms defined pre-launch and refined post-launch by operations personnel. While carrying out these functions on an event-driven basis, the FSW also actively monitors the state-of-charge of batteries. Should power levels fall below acceptable minimums, the flight system will autonomously transition itself (or individual, selected components) to a state or mode of reduced functionality (usually a safemode) with correspondingly lower electrical power demands.

Data Storage and Downlink Bandwidth

Onboard data storage utilization and downlink bandwidth allocation typically fall out of trade studies for ground system operations costs. The ground system will plan its observations to ensure that adequate space is available to store any science data collected during an observation. Similarly, FSW development personnel will design the formats of all telemetry structures to ensure that operations personnel have access to key performance data at required frequencies and to guarantee that customers receive their science data packaged appropriately for ground system processing. However, even in these cases dominated by pre-launch considerations, the flight system has its own autonomous, real-time role to play. Specifically, the FSW must monitor free space availability on the storage device and, in the event of a potential overflow, determine (based on ground-defined algorithms) priorities for data retention and execute procedures regarding future data collection. It also must continuously construct the pre-defined telemetry structures and insert fill data as necessary when data items supposed to be present in the telemetry structure are unavailable. Further, to ensure the necessary link with the ground is maintained to enable successful telemetry downlink, the flight system must appropriately configure

transmitters and orient movable antennas to establish a link with the ground antenna.

Angular Momentum and Propulsion

The last two onboard resources, angular momentum (for reaction wheel-based spacecraft) and propulsion subsystem fuel, can be viewed as physical depletable resources, although the first item more strictly describes the physical behavior or state of the spacecraft. For LEO spacecraft, angular momentum (for reaction wheel-based spacecraft) management typically is fully autonomous and is performed via the interaction of magnetic torquer coils with the geomagnetic field. For orbital geometries where the geomagnetic field strength has diminished below useful levels, excess angular momentum must be dumped via a propulsion system of some sort (hot gas thrusters, cold gas thrusters, or ion jets). Where a propulsion system is utilized to dump angular momentum, often the ground's planning and scheduling system will play a role (even dominate) when angular momentum dumping will occur because of safety concerns regarding autonomous thruster commanding. However even for missions following this conservative operational philosophy, there often will be a contingency mode/state in which autonomous angular-momentum reduction via thruster firing is enabled to deal with inflight anomalies jeopardizing the control and safety of the spacecraft. For spacecraft not using reaction wheels (for example, the future Laser Interferometer Space Antenna (LISA)), angular momentum management is not an issue.

By contrast, management of thruster fuel resources is traditionally almost exclusively a ground responsibility. This allocation of functionality historically has been due to the mathematical complexity of orbit maneuver planning and the earlier limited computational power of OBCs. So, if the planning of orbit maneuvers, the activity expending the bulk of the onboard fuel supply, is a province of the ground system, then management of the propulsion subsystem's fuel budget quite logically would belong on the ground as well. Recently however, considerable interest has been generated regarding the feasibility of autonomous performance of spacecraft orbit stationkeeping activities. In its more elaborate form, autonomous orbit stationkeeping may even be performed in support of maintenance of a spacecraft constellation, coordinating the orbital motions of several independent spacecraft to achieve a common goal, also referred to as formation flying. For these applications where planning and scheduling of the orbit maneuvering function itself is moved onboard, migrating management of the fuel resources to the flight system will be necessary as well.

Flight Autonomy Enablers of Health & Safety Maintenance

Although each spacecraft has ideally been designed to support completion of its assigned science program within its nominal mission lifetime, unpredictable, potentially damaging events threatening termination of spacecraft

operations inevitably will occur sporadically throughout the course of the mission. Many of these events will develop so quickly that by the time the ground recognizes the onset of the threat, develops a solution, and initiates a response, conditions will have worsened to the point that loss of the spacecraft is unavoidable. To deal with these highly dangerous potential problems, as well as a host of lesser anomalies, the flight system is provided with an autonomous fault detection and correction (FDC) capability.

The first responsibility of the flight system's FDC is to monitor ongoing spacecraft function. To this end, FSW analysis personnel in conjunction with systems engineers and operations personnel identify a rather large number of hardware output items and FSW-computed parameters that together provide a thorough description of the state of the spacecraft. The FSW then samples these values periodically and compares them to nominal and/or required values given the spacecraft operational mode. This comparison may be achieved via simple rules and limit checks, or by running models associated with a state-based system. Regardless of the sophistication of the approach, the result of the procedure will either be a "clean bill of health" for the spacecraft or identification of some element not performing within a nominal envelope.

After identification of the existence of a potential problem, the FSW then autonomously commands an appropriate corrective response. The elaborateness and completeness of the corrective response varies depending both on the nature of the problem and degree of independence a given mission is willing to allocate to the FSW. Ideally, the level of response would be a precisely targeted correction that immediately restores the spacecraft to nominal function allowing continuance of ongoing science observations. Usually, a complete solution of this sort will be possible only for minor anomalies, or significant hardware problems where an autonomous switch to a redundant component or a transition to an appropriate FSW state may be performed without incurring additional risk. However, for most major inflight problems, the flight system's responsibility is less ambitious. It is usually not tasked with solving the problem, but simply placing the spacecraft in a stable, protected configuration, for example, transitioning the spacecraft (or an SI) to safemode. The spacecraft then remains in this state while ground personnel analyze the problem and develop and test a solution. Once this process has been completed, the flight system is "told" what its job is with respect to implementing the solution, and then proceeds again with conducting the mission once the solution has been installed.

3.1.2 Satisfying Spacecraft Infrastructure Needs

In addition to its direct, active role in achieving overall mission objectives, FSW has a key role to play as the middle-man between the ground system and the spacecraft hardware. To serve effectively in this capacity, the FSW must provide a user-friendly but secure command structure enabling the ground system to make requests of the spacecraft that will be carried out precisely,

and yet will not simply be acted on mindlessly in a manner that might put the spacecraft at risk. Further, to ensure the spacecraft is capable of responding to these requests in a timely manner, the FSW performs those routine functions necessary to keep the spacecraft available and in near-nominal operational condition.

These functions may be thought of generically as the spacecraft's autonomic system, in much the way that one views a human's respiratory and circulatory system. Since they are so common from spacecraft to spacecraft, and so essential to spacecraft operations, they ironically are often neglected in discussions of spacecraft autonomy. In the following sections, the various elements of this spacecraft operational infrastructure will be described in more detail. Some of this discussion will be a bit redundant with material presented in Section 3.1.1, which dealt with how the FSW enables satisfaction of spacecraft mission objectives. Such material is repeated here more in the context of what the flight system must do in order to keep the spacecraft available and responsive to the ground's needs, as opposed to what it does to accomplish what the ground wants done.

Flight Autonomy Enablers of Command Execution: Validation

Earlier, the various stored command types (absolute-timed, relative-timed, and conditional) were described to illustrate how the FSW is able to execute ground requests faithfully, while still exploiting realtime information that was unavailable to the ground system at the time their requests were generated and uplinked—so as to provide a value-added response to the ground's needs. However, to make safe and reliable use of these command structures, the FSW independently validates commands on receipt, validates them again on execution, and monitors their passage through the C&DH subsystem as they make their way to their local destination for execution.

The first step in the process is to verify that a command (or a set of commands) has not been garbled. For this purpose, when the C&DH subsystem first receives a command packet, the C&DH checks the bit pattern in the header and verifies it matches the expected pattern. At the same time, the C&DH examines the command packet at a high level to make sure it recognizes the packet as something an onboard application could execute. Second, when it is time for the command to be executed, the C&DH determines to which application the command should be sent, and ships the command out to be loaded into that application's command buffer. The C&DH then looks for a message verifying that the command was successfully loaded into the buffer (i.e., that there was room in the buffer for the command). Third, as the application works its way through the buffer contents, it examines the contents of the individual commands to verify they are valid. It also will check the command itself to verify there are no inherent conflicts to executing that kind of command given the current spacecraft state. Finally, once the application determines that the command may be executed, it carries out its prescribed

function or launches it towards its final destination for execution and verifies that it then executes successfully. This multiple-tiered validation process ensures that only valid commands are executed, that they reach their proper destination, and that they're executed properly once they get there.

Flight Autonomy Enablers of Command Execution: Request Orchestration

To this point, discussion of commanding has focused largely on the execution of stored ground requests, primarily relating directly to carrying out the science observing program. However, the FSW must deal not only with this class of commands, but also with realtime ground requests and with commands dealing with engineering and/or housekeeping activities, in many cases originating autonomously from within the FSW itself. In the past, although the FSW was provided with sufficient "intelligence" to keep commands from these various sources from "bumping into" each other, much of this complexity could be managed by the ground. This was especially true for missions where the spacecraft primarily executed absolute-timed stored commands, or for missions like the International Ultraviolet Explorer (IUE) where continuous ground contact (enabled by a Geostationary orbit) allowed the ground to conduct observations via block scheduling and realtime commanding.

However, as more spacecraft take advantage of the benign space environment of earth-sun Lagrange points (e.g., the proposed James Webb Space Telescope (JWST)), the opportunities presented by event-driven commanding orchestrated by a more autonomous onboard scheduling system likely will be increasingly exploited. This in turn will place a greater responsibility on the flight system to manage and prioritize commands from these various sources to optimize science data gathering capabilities without jeopardizing spacecraft health and safety.

Flight Autonomy Enablers of Efficient Resource Management

When the ground system generates the spacecraft's science observation schedule, it implicitly makes a series of assumptions regarding the spacecraft state, e.g., that sufficient power is available to operate the hardware required for the observations, that sufficient angular momentum capacity is present to enable the spacecraft to be oriented properly to observe the targets, that enough science data has been downlinked from onboard storage to permit the storage of newly captured science data, etc. These considerations have already been discussed in some detail in Section 3.1.1, however, it is worth repeating here simply from the standpoint that some elements of resource management (e.g., computing power and internal data transfer) are so intimately associated with the running of the FSW that they have effectively become part of the spacecraft infrastructure. So, for these resources, one tends to view the job of resource management not as an independent application running on

the FSW, but instead as an element of the FSW facilitating the running of applications.

Flight Autonomy Enablers of Health and Safety Maintenance

Previously the FDC capability was described at a high level to illustrate how FSW autonomy is utilized to mitigate health and safety risks that might otherwise lead to onboard failures that could in turn result in failure to achieve spacecraft mission objectives. However, FDC can also be viewed as a key component of the spacecraft infrastructure dedicated to maintaining the spacecraft in a suitable state so the ground can schedule its science observations with confidence, with the knowledge that the FSW will be capable of carrying out its directives effectively, reliably, and safely. As with the case of onboard resource management, there are reasonable arguments for viewing FSW FDC capability both as an enabler of achieving mission objectives and as a critical component of the spacecraft infrastructure.

The most important safety check for all spacecraft is to verify that electrical power capacity is adequate to keep the spacecraft alive. Detection of unacceptably low power levels will engender the autonomous commanding of major load-shedding and (usually) transition to safemode for the spacecraft and its SIs. Verifying that no violations of thermal limits have occurred is almost of equal importance as the power checks. Such violations at best may lead to irretrievably degraded science data, and at worst loss of the SI, or even the spacecraft itself due to potentially irreversible hardware failures such as freezing of thruster propellant lines. At a more local level, celestial pointers in particular are always very concerned with possible damage to their imaging system and/or SIs due to exposure to bright objects or excessive radiation. Another potentially lethal problem is loss of attitude control. Maintenance of attitude control must be checked both to ensure that the spacecraft is able to acquire and collect data from its science targets, and more importantly to ensure that none of the previously discussed constraints (power, thermal, and bright object avoidance) are violated.

Note that not all these safety checks are performed exclusively onboard. The ground system will normally attempt to ensure that none of its commands knowingly violate any of the constraints described above, and the ground system as well will use telemetered engineering data to monitor the spacecraft state to detect any violations that may have occurred. In practice, the crucial job of maintaining the spacecraft system is distributed between flight and ground systems, but when an offending event occurs in realtime onboard, it is primarily the responsibility of the flight system to be the first to recognize the advent of a problem and to take the initial (although not necessarily definitive) steps to solve the problem.

3.1.3 Satisfying Operations Staff needs

Unlike more mundane earth-bound situations where users typically are permitted direct, physical access to the hardware they are using, for spacecraft applications the users effectively interface with the spacecraft exclusively through the FSW, with the exception of the cases of some classes of emergency conditions and some special applications (for example, in a dire emergency, there is a special stream of bits that can be radiated to the spacecraft, and this stream of bits will be intercepted by a special onboard hardware circuit before the data is sent to the onboard computer, and this hardware circuit, when it sees this unique stream of bits in the received signal, will cause the onboard computer to perform a hard reset in order to restore the spacecraft to operations). The FSW both provides access to the spacecraft for commanding purposes and provides insight into ongoing spacecraft operations via telemetry. Furthermore, because the FSW "sees" in realtime what is happening onboard and can take immediate action in response to what it sees, the FSW often is capable of performing tasks that, if assigned to the ground system and FOT, would be much more expensive to do. These cost savings may arise from lower software costs due to simpler modeling requirements onboard or from effective replacement of human staff hours with FSW functionality. Additionally, where those reductions pertain to replacement of repetitive FOT manual activities, one obtains a cost savings multiplier over the entire mission duration. In the following subsections, each of these three services to the operations team will be discussed in more detail.

Autonomy Enablers of Access to Spacecraft Systems

FSW is the mechanism enabling nearly all access to the spacecraft by the FOT. The FSW provides a commanding infrastructure that translates very precisely what the FOT wants done into appropriate hardware and/or software commands, as discussed in previous sections. By this means, the FSW to a certain degree provides the FOT "hands-on" capability with respect to the various spacecraft hardware elements and subsystems. However, while it creates FOT access to spacecraft systems, the commanding infrastructure (through its validation capabilities) protects the spacecraft from the occasional operational errors that might otherwise lead to irreparable damage to delicate hardware.

The FSW even allows the FOT to modify the operation of FSW functions by changing the values of the key parameters that drive the functions' models. For example, most spacecraft model their own orbital position, and the positions of other spacecraft, such as Tracking and Data Relay Satellites (TDRS), through the use of onboard orbit propagators. The starting position and velocity from which future positions are calculated can be specified by the ground in a table, updates to which are uplinked as frequently as required in order to maintain ephemeris accuracy requirements. For other missions, ephemeris updates are performed via command structures rather than tables, but the

basic process is effectively the same. Also, as a protection against missing a routine ephemeris update (either due to inflight problems or simply a ground operations error), FSW for the Medium-Class Explorers (MIDEX) missions notify the ground if the onboard parameters have grown "stale", eventually causing the FSW to terminate that ephemeris model's processing. And if an erroneous update is made, a continuity check in the FSW (implemented on most missions) can detect the mistake, allowing the FSW to reject the update and continue using the existing parameters until an accurate set is supplied by the ground.

Specifically with respect to missions developed by NASA Goddard, the Telemetry Monitor (TMON) (and its more recent variant, the Telemetry & Statistics Monitor (TSM)), capability provides a yet more powerful access into spacecraft function. TMON not only allows the FSW to monitor ongoing spacecraft behavior by checking the values of specific telemetry data points, it also includes a command interpreter program that can execute logical statements and act upon them. As the operation of TMON is driven by uplinked data table loads, TMON can be used to add new onboard functionality without modifying the FSW itself, thereby providing the FOT with a way to work around onboard hardware or software problems in a manner less demanding on the FSW maintenance team.

In addition to supplying a means to influence the spacecraft and FSW behavior, the ground can indirectly change the FSW itself. For small or localized changes, a FSW maintenance development team can patch the FSW, uplinking new code that effectively bypasses some existing code elements and substitutes modified, or even entirely new, functionality in its place. If the number of patches becomes excessive, or if the scale of the upgrade is extremely large, a new version of the FSW program may be developed by the maintenance team and uplinked, which in turn may be patched as future changes are required. For long duration missions having extensive cruise durations (e.g., Jet Propulsion Lab (JPL) missions that may take years to get on station), the cruise phase often is used to re-write the flight code to compensate for major hardware anomalies experienced after launch, or even to complete development and testing of major functionality not finished prior to launch. By these means (command infrastructure, table modification/command uplink, TMON/TSM, and actual FSW coding changes), the FSW affords the ground a remarkably high capacity for accessing, influencing, and modifying spacecraft behavior in flight.

Flight Autonomy Enablers of Insight into Spacecraft Systems

To support the safe and effective utilization of the access to spacecraft systems afforded by FSW, the FSW must also allow the ground a comparably high level of insight into ongoing spacecraft operations. To this end, spacecraft typically are designed so as to ensure that, catastrophic failures aside, the ground will always receive at the very least some minimal level of health

and safety telemetry that summarizes the current spacecraft state, along with significant error messages describing what, if anything, has gone wrong.

When operating in a more nominal state, the spacecraft regularly supplies the ground with fairly detailed information concerning the operational behavior of the spacecraft and its various hardware components. Additional telemetry fields are reserved for FSW-calculated products (and FSW intermediate calculation parameters) deemed to be of value by the FOT and other analytical support staff. As engineering and housekeeping downlink bandwidth typically is in short supply due to optimization of communication hardware trades relative to weight and cost, the allocation of telemetry slots and the frequency with which they are reported is usually a rather difficult process as individual subsystem information needs are traded to avoid exceeding communication bandwidth limits.

However, in the event of an inflight anomaly or the scheduling of a special activity, such as an instrument calibration, at times the nominally optimized engineering/housekeeping telemetry contents may not be adequate to support the immediate needs of the ground. In such cases, the FOT can utilize a capability provided by the FSW to modify telemetry contents. In the past, this was achieved by "flying" several pre-defined telemetry formats, with the current one in use being that selected by the ground. In response to changing inflight conditions, the ground could simply command the use of a different telemetry format more appropriate to current needs. The weakness of this approach lay in its inflexibility, i.e., a telemetry format needed to be defined and be onboard already for it to be used. So if some unexpected conditions occurred requiring a different allocation of telemetry slots and frequencies than that supported onboard, not much could be done immediately to address the problem.

By contrast, the filter table approach utilized by recent spacecraft affords much more flexibility. In principle, any desired presentation of telemetry-accessible data points can be achieved. Of course, given a limited bandwidth, increasing the frequency of a given telemetry item can crowd out other important data items, so a careful trade must always be performed before changing filter table settings. However, at least the higher degree of flexibility ensures that any single FSW-accessible data point can, in principle, be viewed by the ground as frequently as desired.

Flight Autonomy Enablers of Lifecycle Cost Minimization

As budgets for overall lifecycle costs steadily decrease, mission planners increasingly look to FSW as a means to reduce continuing costs of operation. Many current onboard autonomous capabilities, although originally implemented to satisfy mission-specific objectives or to promote enhanced spacecraft H&S, in fact also reduce the work load of operations personnel, in turn enabling the reduction of staffing levels without the loss of efficiency or increased risk. For example, for decades, spacecraft have autonomously main-

tained attitude control, calibrated gyro-drift biases, propagated their orbital ephemeris, managed battery charging, maintained thermal constraints, packaged and stored science and engineering data, checked for limit violations, and (on detection of any violations) either fixed the problem or placed the spacecraft (or localized element) in a benign state pending ground attention. Trying to perform any of these functions with ground personnel "in the loop" would not only be less efficient and less safe, it would also be far more expensive than delegating the responsibility to the flight system.

Recently, more elaborate flight-autonomy capabilities have been introduced specifically to reduce operational costs. For example, to break linkages between spacecraft target-pointing and communications (antenna pointing), the Rossi X-ray Timing Explorer (RXTE) mission introduced an autonomous antenna-manager function responsible for selecting the appropriate high gain antenna (HGA) to use compatible with the current spacecraft attitude. This capability not only supported greatly reduced lead times on changing targets to observe a TOO, but also reduced staff efforts (and costs) in scheduling TDRS contact times by eliminating couplings between TDRS scheduling and onboard antenna selection, which often is a factor when optimizing communications contact time.

And for JWST, use of an onboard event-based scheduler could reduce overhead time (in turn raising observing efficiency) and reduce both ground-system modeling costs as well as the need for spacecraft "hand holding". And further in the future, increased onboard processing of science data may not only enable increased capabilities to exploit TOOs detected in real time onboard, it could for some missions also permit a reduced science data downlink volume, with associated operations cost reductions, as the science community gains confidence in the accuracy and reliability on the onboard processed product.

It should be noted that these reductions in operations costs do not themselves come without a cost. The development of new FSW functionality typically is an expensive undertaking, both from the standpoint of coding the new capability and from the standpoint of the testing required to ensure that no inadvertent harm is done to the spacecraft. The impact of these software costs however are lessened when the new autonomous function is implemented for a long duration mission where the costs can now be traded relative to the, say, 10 years of operational effort that the FSW replaces. Similarly, when several missions can use the new capability, the up-front development costs for the first mission can be seen as a long term investment yielding savings both on that mission and downstream missions. Hopefully as the expense of developing FSW continues to decline, and as greater FSW reuse becomes possible, the trade of continuing operations costs for new FSW autonomy will be an increasingly favorable one.

3.2 Brief History of Existing Flight Autonomy Capabilities

In the previous sections, the reasons for developing flight autonomy, and the flight autonomy capabilities that were developed in response to those needs, were discussed in some detail. In the following sections, those flight autonomy capabilities will be grouped in accordance with the general time periods in which they were developed at GSFC and examined relative to the contributions they made for specific spacecraft on which the flight software functionality was flown. General time periods reviewed are the 1970's, 1980's, 1990's, and 2000's.

3.2.1 1970s and Prior Spacecraft

During the 1970's, NASA made the first attempt to standardize onboard flight data systems with the creation of the NASA Standard Spacecraft Computer (NSSC), versions I and II. The NSSC-I, a derivative of that flown on the Orbiting Astronomical Observatory-3 (OAO-3) in 1972 and IUE in 1978, was first flown on the Solar Maximum Mission (SMM) in 1980 (originally scheduled for launch in 1979). Compared to modern flight computers, the NSSC-I was slow, had very limited memory, was cumbersome when performing mathematical functions due to its small word size and lack of floating point arithmetic, and was awkward to program due to the exclusive use of assembly language. However, it was extremely reliable and was used successfully to support the onboard needs of many missions, from SMM in 1980 to the HST payload in 1990.

The NSSC computers, and other OBCs with comparable capabilities, such as those used on the High Energy Astronomical Observatory (HEAO) series, were employed successfully in the 1970's to support a basic foundation of spacecraft autonomy, including stored commanding, telemetry generation, FDC, orbit propagation, and pointing control. Stored commanding capabilities included the (now) standard set of absolute-timed, relative-timed, and conditional commands, as discussed previously. A degree of FDC (for constraints such as bright object avoidance and minimum power levels) also was present in the form of limit checks on key onboard sensor measurements, with autonomous mode transition capabilities to familiar safemodes, such as Sunpoint.

In the area of pointing control, for the case of HEAO-2, pointing control and ground attitude determination accuracy (using star tracker data) were required to be good to 1 arcminute and 0.1 arcminute, respectively—a very demanding requirement for the time. And the Solar Maximum Mission's (SMM) fine Sun sensor was so well calibrated that its attitude could be controlled autonomously to 5 arcseconds with respect to the Sunline. Attitude control was achieved using onboard closed-loop proportional-integral-derivative (PID) control laws (including feed-forward specification of environmental torques)

and Kalman filters (for optimization of attitude-error determination and gyro-drift bias calibration). So fundamentally, the control approaches used on these spacecraft from the 1970s were quite similar to those currently used on modern spacecraft in support of spacecraft slews, target acquisition, angular momentum management, and maintenance of attitude during science observations.

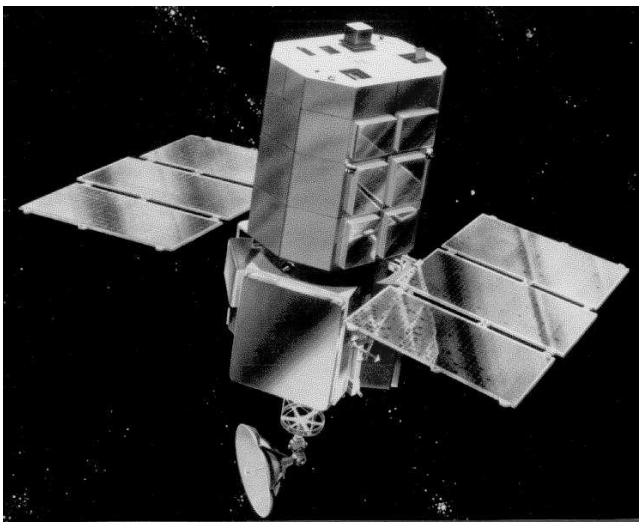

Fig. 3.1. Solar Maximum Mission (Image Credit: NASA).

Originally SMM also possessed the rudiments of an autonomous target identification and acquisition capability that presaged the more elaborate capability implemented in HST (as discussed later), which in turn was a next step on the road to a fully autonomous TOO response capability. Specifically, when SMM's SI detected a Solar flare, the data was processed onboard, the flare's location was determined, and the spacecraft was autonomously reoriented to observe the phenomenon. This feature enabled a far quicker response than would have been the case if the data processing and commanding responsibility resided in the ground system, allowing time-critical measurements to be made during the early stages of the flare's duration. The Orbiting Solar Observatory-8 (OSO-8), launched in 1975, also could steer its payload platform independently to expedite acquisition of its short-lived targets.

The evolving nature of flight autonomy can be seen within the decade of the 1970's itself just by noting the significant increase in pointing independence between HEAO-1 and HEAO-2. Specifically, HEAO-1 (launched in 1977) relied on the ground to provide it periodic attitude reference updates (every 12 hours) based on ground attitude determination. Just two years later, HEAO-2

already possessed the capability to compute its own attitude reference update, based on ground-supplied guide-star reference information, a capability also implemented in SMM's ACS for autonomous control of roll about the Sunline. Furthermore, HEAO-2 could autonomously sequence through a weekly target list, adjusting the order of the targets so as to economize on use of thruster fuel used in momentum dumping, a remarkable degree of independence even relative to the 1990s missions discussed later.

Examination of FDC also shows a dynamic quality. Although many spacecraft flew hard-coded limit checking and response code, SMM's statistical monitor performance function provided additional flexibility. It allowed operations personnel to specify additional FSW parameters to be monitored autonomously onboard beyond those specified in the at-launch flight code, without making a modification to the code itself. This function was itself improved upon in the 1980's with the introduction of TMON, a telemetry monitor that also supported an autonomous onboard response capability.

3.2.2 1980s Spacecraft

Relative to the 1970's, the period of the 1980's saw the launch of larger, more expensive, and more sophisticated spacecraft. Many of these spacecraft (e.g., HST and the Compton Gamma Ray Observatory (CGRO)) were supposed to launch in the mid- to late-1980's, but in actuality launched in 1990 because of delays due to the loss of Challenger.

In the case of HST, a more powerful OBC (the DF224) enabled the development of more elaborate pointing-related mathematical algorithms as well as a wider variety of safemode options (supported by a larger number of FDC checks) than had been present on previous spacecraft. In particular, use of HST's fine guidance sensors (FGSs) required the development of rather complex (by onboard standards) mathematical algorithms to command the FGSs and process their data. In fact, the processing demands of the FGS functionality was so high that the HST FSW's 10 hertz processing rate was created for and exclusively dedicated to this purpose. The FGS guide-star acquisition algorithms were themselves extremely powerful, exploiting the full command-construct repertoire to achieve the intricate branching/looping logic needed to optimize the probabilities for acquiring the guide stars essential for performing HST's science.

The very existence of the 10 hertz processing rate points to an additional noteworthy aspect of HST's autonomy capabilities that often is taken for granted, namely its executive function. For the sake of simplicity, most FSW development efforts try to limit the number of tasks to as few as possible, usually one or two. However, because of HST's unique computational, precision, and timing demands, the HST pointing-control subsystem (PCS) software required five distinct processing rates, namely, 1000 hertz for the executive, 40 hertz for primary PCS control laws and gyro processing, 10 hertz for FGS processing, 1 hertz for star tracker processing, ephemeris modeling, and FDC,

and 1/300 hertz for the minimum energy momentum management control law. Just the management of these very different, and often competing, tasks demonstrated a significant degree of executive autonomy.

HST's FSW also displayed a high level of autonomy in acquiring science targets through "conversations" between the NSSC-I computer supporting SI commanding and processing, and the DF224 computer responsible for spacecraft platform functionality. For example, for science observations where the target direction was not known to a sufficient level of accuracy to guarantee acquisition in an SI's narrow FOV, the DF224 could initiate (through stored commanding) a small scan of the region of the sky surrounding the estimated target coordinates. The passing of the target through the FOV of the SI would then trigger a sudden increase in SI intensity measurements, which would then be noted by the NSSC-I. On completion of the scan, the NSSC-I could then request (through a limited interface between it and the DF224) that the spacecraft attitude be returned to that pointing. The DF2244 would then create and initiate a realtime slew command satisfying the NSSC-I's attitude change. As with the SMM autonomous target acquisition feature, HST's capability provided a far quicker response than would have been the case had data processing and commanding responsibility resided in the ground system, a very important consideration given the high cost of HST operations and the extraordinary demand for HST observing time.

By contrast, because of its lower pointing accuracy requirements, the Extreme Ultraviolet Explorer (EUVE), also originally scheduled for the 1980s but actually launched in 1991, did not require a level of sophistication in its ACS subsystem as high as that required on HST. However, it did provide a higher level of flexibility with respect to onboard data monitoring and limit checking. EUVE's telemetry monitoring capability (referred to as TMON, and also flown on CGRO and the Upper Atmosphere Research Satellite (UARS)) permitted the user to select, after launch, specific data points to monitor. In addition, TMON provided a limited logic capability to respond onboard to detected spacecraft conditions (observed via limit checks on data or flag checking) through autonomous generation of commands. EUVE's FSW also included a separate statistics monitor program (called SMP) that was later combined with TMON and flown on MIDEX spacecraft as the TSM program (see Section 3.1.3). Finally, EUVE possessed an extremely user-friendly table-driven limit-checking/response system that has served as the model for later missions in the explorer series.

As an early predecessor of true event-driven operations, EUVE utilized an Orbit Time Processor (a table-driven task) that allowed its FSW to define orbit-based events, a variation on the relative-time-based commanding discussed earlier. Occurrence of the event could then trigger a relative time sequence (RTS), a task, or set an event flag that in turn could be monitored by a running task. The EUVE FOT employed this enhancement to the standard stored commanding infrastructure to re-phase the timing of EUVE's survey mode, which operated within a third of an orbit duty cycle. EUVE also used

its Orbit Time Processor to send dusk/dawn commands to the survey instrument.

Although HST's FDC capabilities are not as flexible as EUVE's, HST checks for a much wider spectrum of anomalous conditions, with a larger range of autonomous responses. For example, HST provides four distinct software safemodes: inertial hold, a multi-staged Sunpoint, zero-gyro (derived from Sunpoint), and spin-stabilized. Also, in response to guide-star reacquisition problems associated with radiation hits on its Fine Guidance Electronics (FGE) following SAA entrances, HST's FSW developers have implemented an FGE memory-refresh function that restores key FGS commanding parameters to their latest values prior to the SAA entrance. Note that many of HST's FDC capabilities were added post-launch in response to problems experienced inflight, which not only illustrates the power of FSW to solve unanticipated operational problems that can be dealt with no other way, but also provides a strong argument in favor of creating and maintaining a strong FSW maintenance team capable of responding to those operational problems when they do occur.

3.2.3 1990s Spacecraft

Relative to the 1970's and 1980's, the 1990's witnessed major hardware and infrastructure advances that enabled greater onboard capabilities. The flight computers were more powerful, with larger memories, and were faster, enabling more sophisticated algorithms and models. Floating point arithmetic and higher level languages (such as C, C++, and Ada) allowed FSW code to be written more like comparable ground system code. For example, object-oriented design concepts can now be used to make flight code more re-usable, and in the long run potentially cheaper. Thanks to high capacity, lightweight, and cheap solid state storage devices, larger amounts of science data may be stored onboard and packaged more conveniently (with respect to end-user needs) without undue concern for added overhead space costs (although in practice this gain has been largely offset by corresponding increases in SI output data volume). More sophisticated operating systems are now available to handle the masses of data and manage the more elaborate computations. The cumulative result of this technological progress has been to enable a series of new individual flight autonomy capabilities targeted to the needs of specific missions, as well as to support the development of entirely new FSW concepts.

To meet demanding time requirements for TOO response, the Rossi X-ray Timing Explorer (RXTE) FSW included three new flight autonomy capabilities: onboard target quaternion computation, target quaternion validation, and the antenna manager. The first two enable a science user to specify simply the target's right ascension and declination, and whether there are any special roll coordinate needs. The FSW then takes this targeting information expressed in the natural "language" of the user, transforms it appropriately (i.e., into quaternion format) for use in slewing to and acquiring the target,

quality-assures the attitude versus Sun-angle avoidance, and then slews the spacecraft to point to the target at the ground-specified time.

A new RXTE autonomy capability was proposed, but not funded, that would have greatly enhanced RXTE's already superb TOO response time. If RXTE's All Sky Monitor (ASM) detected the signature of a possible gamma ray burster (GRB), the ASM FSW could have determined the celestial coordinates of the potential TOO. After verifying that those coordinates had not previously been observed, the ASM could then have communicated the GRB celestial coordinates to the OBC, which could then have utilized RXTE's existing capabilities to compute and validate the new target quaternion. Next, the FSW could have autonomously determined the right time to break away from currently scheduled observations, slew to that target, and then generate the appropriate SI configuration commands so observations by RXTE's Proportional Counter Array (PCA) could be made. Finally, using onboard SAA contour information and spacecraft ephemeris, the FSW could have determined the time at which those PCA observations could commence relative to earth occultation and SAA entrance times. Although this capability was not flown on RXTE, it has been implemented successfully and is the key to satisfying the rapid TOO response time requirement, for the Swift mission (see Section 3.2.4).

The third item among the list of autonomy features implemented pre-launch—the antenna manager—enabled de-coupling of science and communications scheduling. For missions where HGA selection is pre-planned by the ground, a change in target attitude due to inclusion of a TOO can induce changes in HGA commanding for that target observation period and even future target observation periods downstream. Further, for ground algorithms that optimize TDRS switchover times and HGA selection choices in order to maximize total TDRS contact time, relatively small changes in the attitude profile can cause major changes in desired TDRS schedule. By contrast, RXTE's antenna manager allowed the FSW to determine in realtime which was the best HGA to use to close the link with the scheduled TDRS spacecraft, based on the FSW's realtime knowledge of its attitude and the relative orbital positions of RXTE and TDRS (derived from onboard orbit models). Also, knowing the TDRS schedule, RXTE's FSW could autonomously determine when HGA transmitters should be turned on and when playbacks from the solid state recorder (SSR) should start. These autonomous features were used routinely with great success until a transponder failure eliminated the two-HGA capability.

RXTE and the Tropical Rainfall Mapping Mission (TRMM) also provided enhanced flexibility in telemetry formatting via their telemetry filter tables. In practice, the same amount of planning effort (the most laborious part of the job) would be required to make major changes to its standard telemetry configurations (identified as operationally necessary pre-launch) as would be the case for earlier approaches to telemetry formatting, but once determined,

the modified telemetry formatting could be implemented via a simple table uplink as opposed to a FSW change.

RXTE and TRMM also flew a more sophisticated version of TMON, called TSM (originally developed for the Solar Anomalous and Magnetospheric Particle Explorer (SAMPEX) mission), that supported all the functionality of TMON, i.e., monitoring telemetry points, performing limit checking, and initiating stored command sequences and associated event messages on limit failure. In addition, TSM maintained statistical data for each monitor point and accepted FSW reconfiguration commands. Statistical information generated includes telemetry-point current value and time, minimum and maximum values seen with associated times, average value, and number of times the monitor point has been seen. TSM was particularly useful to the RXTE mission as a means to deal with star tracker problems experienced inflight without having to implement major changes in the flight code itself.

Also in the area of diagnostic functionality, Landsat-7 (launched in 1999) experimented with a high level FDC capability, in addition to flying a more traditional one. The FDC functions for most spacecraft have a one-to-one quality, i.e., a trigger is received and an associated response is executed. The potential problem with this approach is that a higher level problem (for example, a failure in the ACE) can corrupt output data from several ACS sensors that could be misinterpreted as the individual failures of all those ACS sensors, potentially resulting in unnecessary autonomous switches to their redundant components. To protect against a problem of this kind, Landsat-7 implemented a Boolean logic function that would examine all error flags generated at a component level and, by comparing the error flag pattern to a set of patterns maintained onboard defining the signature of higher level problems, deduce the true cause of the current anomaly and respond accordingly. By setting the counters associated with the triggers for the higher level failures to lower values than for the counters associated with the component level failures, the Landsat-7 FSW would be able to switch out the higher level hardware element before the cascade to redundant components commenced.

At a somewhat more detailed level, SAMPEX incorporated a new autonomous calibration function, which also has been flown on other spacecraft in the Small Explorer (SMEX) series. SAMPEX possesses the capability to calibrate its magnetometer coupling constants (relative to the magnetic torquer bars) inflight, relieving the FOT of the burden of collecting the necessary engineering data, processing it, and uplinking the modified calibration parameters to the spacecraft.

Lastly, some very interesting new ideas have been implemented at JPL. Because of the long cruise periods until their spacecraft achieve their mission orbits or swing-bys, JPL has the luxury of experimenting with their FSW after launch and even making wholesale changes (or simply completing the original coding effort) after launch. Their deep space missions also by their very nature may require more autonomy than is typical of GSFC missions. Because of the long communications-delay times inherent in a deep space mission and because

of the time critical aspects associated with celestial flybys, JPL has been experimenting with autonomous target identification and acquisition functions that are more elaborate than those flown at GSFC. At a more fundamental structural level, the New Millennium Program's Deep Space One (DS1) FSW was initially designed with Remote Agents having responsibility for multi-task management, planning and scheduling, and model-based FDC [100, 109]. In practice (due to schedule conflicts), the mission was flown using a more conventional FSW implementation, but the Remote Agent-based version was activated briefly for test purposes.

3.2.4 Current Spacecraft

A number of interesting new autonomy capabilities have been flown on GSFC spacecraft launched in the 2000's. First, the recent development of quaternion star trackers has provided spacecraft like the Willsinson Microwave Anisotropy Probe (WMAP) (launched in 2001) a true "Lost in Space" capability. Previously, the limited star catalogues flown on GSFC spacecraft, along with the simple star identification algorithms utilized in the FSW, required that fairly accurate *a priori* spacecraft attitude knowledge be available onboard for reliable fine attitude updates to be performed. Now however, star trackers are available that use more extensive internal catalogues and more powerful star identification algorithms to provide quaternion information to the FSW without previous attitude knowledge. This new autonomy capability will both support ongoing science observing and streamline recovery from safemode entry. These new star trackers also output the change in attitude, providing a direct back-up and sanity check to the primary body-rate data supplied by the gyros.

Second, the Earth Observing Spacecraft (EOS) Aqua (formerly EOS-PM, launched in 2002) has implemented an autonomous communication capability referred to as "Call 911". When a serious anomaly occurs on the spacecraft, a stored command sequence reconfigures the communications downlink (from the spacecraft to the TDRS System (TDRSS) to the ground station) and broadcasts an unscheduled Multiple Access (MA) message via the TDRS Demand Access capability. The message is forwarded from White Sands to the EOS Aqua control center, triggering an alarm that unexpected telemetry has been received. The telemetry provides a status message describing the anomaly. The ground can then be ready for contingency commanding at the next scheduled ground contact, or declare an emergency and schedule TDRSS S-band Single Access (SSA) contact time.

Third, on the Swift spacecraft (launched 2004), the key to the rapid target of opportunity response to detected GRBs was a capability considered previously as a post-launch update to the RXTE's FSW (see Section 3.2.3). When Swift's survey instrument (the Burst Alert Telescope (BAT)) detects the signature of a possible GRB, the FSW determines the celestial coordinates of the potential TOO. After verifying that those coordinates had not previously

been observed, the FSW communicates the GRB celestial coordinates to the OBC, which then computes and validates the new target quaternion. The FSW autonomously determines the right time to break away from currently scheduled observations, "swiftly" slews to that target, and generates the appropriate SI configuration commands so high precision observations by Swift's Narrow Field Instruments (NFI) (the X-ray Telescope (XRT) and UV/Optical Telescope (UVOT)) can be made. Swift also (via TDRSS) can respond to TOO alerts identified by other observatories. This new autonomy function is a very significant first step in the direction of "smart" SIs controlling the science mission and all resources required to perform the science mission, as opposed to the traditional operational approach in which the spacecraft/ground system controls the mission and configures the SIs to perform the observations.

Fourth, a capability was considered for the Triana mission (launch postponed indefinitely for budgetary reasons) that would have utilized science data to point Triana's imaging instrument at the earth. The FSW for the Epic SI used to image the earth could have been used to process the science data in order to derive the earth centroid. The centroid data would then be communicated to the spacecraft platform FSW for use in improving the accuracy of the spacecraft's pointing towards the earth center, or a region offset from the center. This basic autonomy capability will, however, fly on the Solar Dynamics Observatory (SDO) (scheduled launch date, December 2008). SDO's guide telescopes (providing precision-pointing support to its science instruments) will supply data to SDO's FSW, which will then compute a Sun centroid to support direct autonomous pointing of SDO's science instruments at the Sun without realtime ground-processing of science data.

Fifth, an experiment in spacecraft formation flying was performed in 2001 using the EO-1 (launched in 2000) mission and the existing Landsat-7 mission. Landsat-7 was a passive participant, simply executing its normal science mission. EO-1, equipped with a GPS receiver to measure the EO-1 orbital coordinates in realtime and an orbit propagator supplying predictive Landsat-7 orbital coordinates, maintained approximately a one minute separation between its orbit and the Landsat-7 orbit. This experiment successfully demonstrated an important capability that can be used by future earth science constellation missions to synchronize science data taken at different local times and to use images gathered by the "lead" spacecraft over the target to optimize science instrument configuration on the trailing spacecraft, or establish for the trailing spacecraft that the target is "socked-in" so that advance preparations for viewing the next target can begin.

3.2.5 Flight Autonomy Capabilities of the Future

Future GSFC missions are expected to advance current onboard capabilities significantly in the areas of planning and scheduling and FDC. JWST (and several other missions currently under development) have proposed the use

of onboard event-driven scheduling to exploit the benign thermal environment of the L2 Lagrange point (Figure 3.2). An Observation Plan Execution (OPE) function would enable the spacecraft to move through its observation schedule on an as-ready basis, rather than pausing to hit absolute time points dictated by traditional fixed-time scheduling approaches. On the other hand, if anomalous conditions occurred that precluded observing the desired target (for example, guide stars not being acquired), the OPE function would simply move on to the next target on the list without further loss of observing time. So use of the OPE function should produce some gains in overall observing efficiency. Further, by taking advantage of realtime knowledge onboard concerning the spacecraft's angular momentum, the OPE function could intelligently plan when to perform necessary angular momentum dumps with minimal impact to science observations.

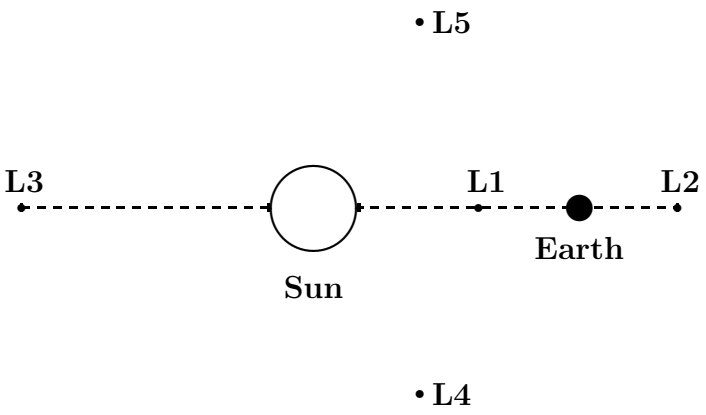

Fig. 3.2. Lagrange Points (not to scale).

Other major enhancements in FSW capabilities are likely to be driven by the needs of the interferometry missions proposed to study star formation, planet formation, etc. In the past, spacecraft were launched mounting relatively smaller science instruments that were pointed at their science targets by pointing the entire spacecraft (e.g., HST). Some science instruments were equipped with swivels that allowed the science instrument to be pointed independently and, in the case of survey SIs, the swivel could be rotated continuously to map out a swath of the sky (e.g., RXTE and Swift). On a few missions, the survey function was carried out without a swivel by continuously

spinning and precessing the entire spacecraft (e.g., WMAP). But for interferometry missions, whose performance capabilities are driven by the length of their baseline, in effect a small spacecraft bus supports a very large science instrument (on the order of many tens of meters long). And for some interferometry missions currently on the drawing boards, to achieve even larger baselines, the science instrument is a composite object that is an amalgam of the individual light collecting capabilities of many individual detector spacecraft whose data is consolidated within a hub spacecraft. For these missions, pre-proposal planning usually concentrates on developing a feasible design for the science instrument, paying less attention to the spacecraft bus, whose design needs are often assumed to be satisfiable by an existing Rapid Spacecraft Development Office (RSDO) spacecraft design. This is a major paradigm change from GSFC's earlier approach of paying equal or greater attention to the spacecraft bus design during the early planning phases.

The most interesting developments in flight autonomy may be those features required to support formation flying and spacecraft constellations. Such missions will demand a much heightened degree of spacecraft self-awareness and self-direction, as well as an awareness of the "outside world". Until now, a spacecraft has needed to be knowledgeable regarding outside "entities" to the extent that it needed to use them. For example, to use a TDRS spacecraft to communicate with the ground, a spacecraft had to know both its own ephemeris and that of the TDRS spacecraft. But for a constellation of spacecraft (for example, for a distributed interferometry mission) to maintain the collective grouping needed to achieve their overall mission objectives, one or more (potentially even all) of the constellation will have to possess key knowledge of all near-neighboring (or possibly all) constellation members in order to synchronize orbital positions, SI configurations, onboard data processing and communication schedules, etc.

The Laser Interferometer Space Antenna (LISA) mission is a particularly high technology example of this kind of constellation mission. LISA, the first space-based attempt to detect gravitational radiation, will consist of three spacecraft maintaining a triangular formation, with each leg of the triangle being 5 million km in length (Figure 3.3). The constellation will be located within the earth's orbit about the Sun, about 20 degrees "behind" the earth. Each spacecraft in the constellation will mount two lasers. Each laser will be directed towards a cube (called a proof mass) floating "drag free" within a containment cell housed within one of the other spacecraft. So each spacecraft mounts two lasers and two cubes, and each triangle leg is formed by two laser beams. The first beam is directed from the master spacecraft to a "slave" spacecraft, which phase locks its laser to the incoming beam and directs its reply back to the master. The master then mixes the incoming light with a small fraction of its original outgoing light to produce an interference pattern that can be processed to determine changes in distance between the free-floating proof masses good to better than the size of an atom. Once software processing has eliminated the many noise effects and other perturbations that can

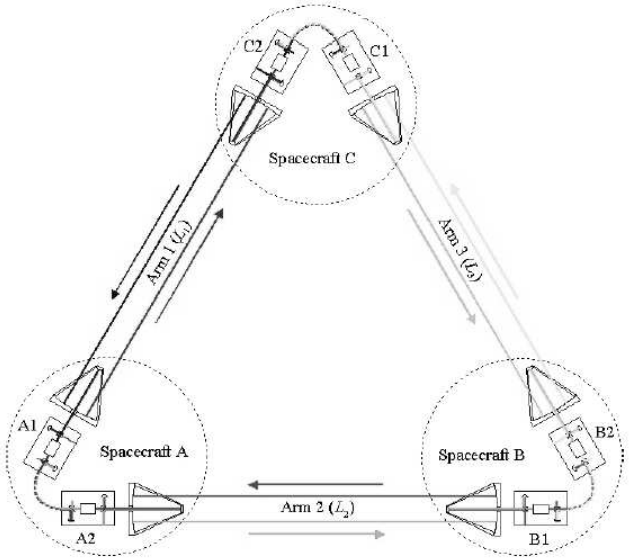

Fig. 3.3. Laser beam exchanges between the three LISA spacecraft.

mask the desired signal, the remaining information can be used to detect the presence of gravity waves (whose existence is predicted by Einstein's General Theory of Relativity, but have not as yet been directly detected), which when passing through the antenna will cause the proof masses to move apart by an amount comparable to the sensitivity of the measuring apparatus.

3.3 Current Levels of Flight Automation/Autonomy

To get a feel for where those new automation/autonomy opportunities may reside, it is useful to associate the items on the previous list of operational activities with rough estimates of the activity's current level of flight autonomy, specifically high, medium, low, or not applicable. The annotated list appears in Table 3.2.

Activity 2, command loading, is the ground activity responsible for uplinking data to the spacecraft. It is already a mostly automated process and within the next 10 years will likely be a fully autonomous ground process. Downlinked data capture and archiving (activities 6 and 10) also have been automated and will probably be fully autonomous ground processes in the next decade. Part of data capture is data validation, which is a necessary part of the flight/ground "conversation" required for onboard science data storage management.

Table 3.2. Operational activities rough estimates of current level of flight autonomy

ACTIVITY	CURRENT FLIGHT AUTOMATION/AUTONOMY LEVEL
1. Planning & Scheduling	low
2. Command Loading	n.a.
3. Science Schedule Execution	medium
4. Science Support Activity Execution	medium
5. Onboard Engineering Support Activities	high
6. Downlinked Data Capture	n.a.
7. Data and Performance Monitoring	medium
8. Fault Diagnosis	low
9. Fault Correction	low
10. Downlinked Data Archiving	n.a.
11. Engineering Data Analysis/Calibration	low
12. Science Data Processing/Calibration	low

As all three of these areas are concerned nearly exclusively with processing data within the ground system itself, there is no real role for the flight system to play in expediting the process directly. However, there are indirect, supporting functions, such as onboard packaging of data prior to downlink and initiation of the communications link between flight and ground where the flight system could play a larger role. Routine onboard operations of this sort are subsumed under activity 5, onboard engineering support activities, as will be discussed later.

The remaining 9 operational areas all offer significant opportunities for an expanded, autonomous flight presence. Those areas labeled "low" (activities 1, 8, 9, 11, and 12) currently are largely ground dominated, but could be at least partially migrated onboard to produce overall system cost and/or efficiency gains.

The "medium" activity areas (3, 4, and 7) already are performed onboard, but either there will be room for expanded functional scope (potentially replacing ground effort) or the ground system typically would have to generate some support products to simplify current onboard processing. So cost/efficiency gains potentially can be realized either by ground-to-flight migration or by introducing entirely new functionality to the flight system. Finally, the activity area 5 labeled "high" is fully autonomous onboard right now, but new functionality could be introduced to produce improved system performance.

4

Ground Autonomy Evolution

Having focused on automation and autonomy in space-based systems in the previous chapter, attention is now directed towards ground-based systems, particularly the automation of spacecraft control centers. We describe a strategy for automating NASA ground-based systems by using a multi-agent system to support ground-based autonomous satellite-subsystem monitoring and report generation supporting mission operations. Over the last several years, work has progressed on developing prototypes of agent-based control centers [2, 36, 87, 134]. With the prototypes has come an improved understanding of the potentials for autonomous ground-based command and control activities that could be realized from the innovative use of agent technologies. Three of the prototypes will be described: AFLOAT, LOGOS and ACT.

4.1 Agent-based Flight Operations Associate

AFLOAT (an Agent-based Flight Operations Associate), was a prototype of a multi-agent system designed to provide automated expert assistance to spacecraft control center personnel. The overall goals of AFLOAT were to prototype and evaluate the effectiveness of agent technology in mission operations.

The technical goals of AFLOAT were to:

1. Develop a robust and complete agent architecture,
2. Address the full spectrum of syntactic and semantic issues associated with agent communication languages,
3. Develop and evaluate approaches for dealing with the dynamics of an agent community which supports collaborative activities,
4. Understand the mechanisms associated with goal-directed activities,
5. Develop a full range of user-agent interface capabilities including the development and use of user modeling techniques to support adaptive user interfaces and interactions.

The following discusses the structure, architecture, behaviors, communication, and collaboration required to coordinate the activities of the agents in supporting unattended mission operations in a satellite control center.

4.1.1 A Basic Agent Model in AFLOAT

An agent is a computer-based autonomous process that is capable of goal-directed activities [17, 45] and can accomplish tasks delegated to it with minimal reliance on human intervention. Because it is goal-directed it can allow the user to specify simply what he/she wants [38], leaving to the agents the how and where to get the information or services. Each agent is also able to participate in cooperative work as an associate with humans or as part of a community of cooperating agents.

The agent architecture used in AFLOAT provided appropriate structural elements and behavior to support basic requirements for adaptive reasoning [172]. An agent is adaptive to the extent that it can respond to short-term and long-term changes in its operational environment, deal with unexpected events and reason opportunistically, maintain a focus of attention among multiple goals, and select and perform appropriate actions.

Structural Elements of an AFLOAT Agent

Each agent in AFLOAT had three structural components (Figure 4.1):

1. An inter-agent communication interface,
2. A monitor, and
3. A knowledge base.

The inter-agent communication interface was responsible for validating the inter-agent semi-structured language format, sending outgoing messages, receiving incoming messages, and broadcasting messages to other agents. The monitor was responsible for monitoring interactions between agents, incoming and outgoing messages, and the state of the agent, and maintaining a history of the agent's actions. An agent's past-actions history supports the agent's learning by drawing from experience when presented with new tasks. Each agent's knowledge base consisted of three elements:

1. A strategist, or a decision-theoretic planner,
2. Problem-specific context descriptor, and
3. A set of procedures or rules for domain-dependent actions.

The strategist was responsible for planning and scheduling the actions that an agent must perform to achieve its goal. The problem context descriptor defined specific attributes of each request to ensure that each agent's attention was focused on the problem at hand. Problem solutions were modeled as domain-dependent procedures. The internal models maintained functions

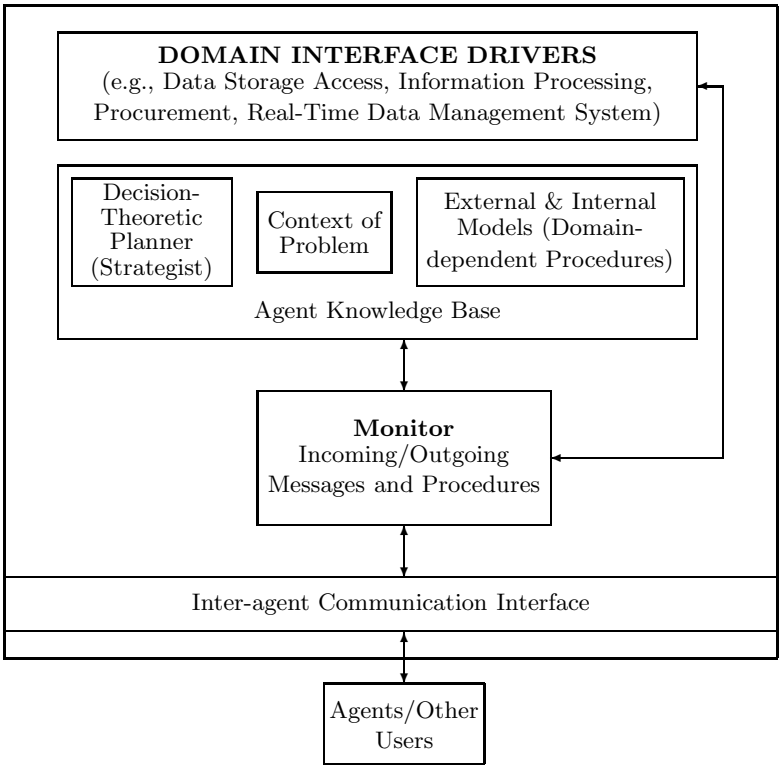

Fig. 4.1. Architecture definition for a domain-independent knowledge-based (a.k.a. deliberative) agent in AFLOAT.

(such as managing access to the skills of each agent or maintaining its message buffer) that were private to each agent and were not accessible to external agents. The external models module maintained global functions that were accessible to other agents. Both types of model were used to maintain a set of actions necessary to achieve the agent's goals. The actions were stored as either rules or procedures. In AFLOAT, the strategist was implemented as a Strategy-Schema [172] which maintained each agent's subgoals for each request it received. The problem context descriptor was modeled as a Context-Schema to hold the features of a specific request, such as attention-focusing information, default knowledge, and standing orders. Request-specific procedures were modeled as Procedure-Schemas.

Behavioral Elements of an Agent in AFLOAT

Each agent in the AFLOAT architecture had six high-level behavior characteristics similar to Laufmann's "action-oriented" attributes [104]. The attributes or capabilities are:

1. Autonomy,
2. Learning,
3. Migration,
4. Persistence,
5. Communication, and
6. Cloning/spawning.

Autonomy/semi-autonomy is the ability of an agent to respond to a dynamic environment without human intervention, thus improving the productivity of the user. When presented with a request, it can use its own strategy to decide how to satisfy the request. Each agent is capable of a type of learning that enables it to more responsively interact with its user community over a period of time. Learning also enables the agent to keep abreast of changes in its operational environment. The dynamic behavior of the agents is triggered by either command or event-driven stimuli. Migration is the ability of an agent to relocate to other nodes to accomplish its tasks. This ability can support load balancing, improve efficiencies of communication, and provide unique services that may not be available at a local node. Persistence is the ability to recover from environmental crashes and support time-extended activities, thus reducing the need for constant polling of the agent's welfare by the user and providing better use of the system's communication bandwidth. A communication ability provides an agent with the mechanisms for supporting agent-agent and user-agent interactions either through an agent communication language or a domain-specific natural language. Spawning is an agent's ability to create other agents to support the parent agent, thereby promoting dynamic parallelism and fault-tolerance. It is our opinion that these capabilities are necessary for building autonomous satellite control centers.

The agent architecture described above is generic enough for use in automating the operations of the control center of any spacecraft. The Explorer Platform's (EP's) [137] satellite control center was selected as a domain to test the feasibility of AFLOAT. Figure 4.2 depicts the interactions between different components of the EP satellite control center. The elements shown in the diagram are similar to those found in a typical satellite operations control facility. A taxonomy of the EP subsystems and data extraction system is also shown in Figure 4.2. The diagram describes the interactions between the physical model (i.e., elements above the mnemonics database), and the logical model (i.e., elements below the mnemonics database) of the components of the EP system.

The main goal of the AFLOAT project was to implement a multi-agent architecture that could interface directly with sources of data from a satellite

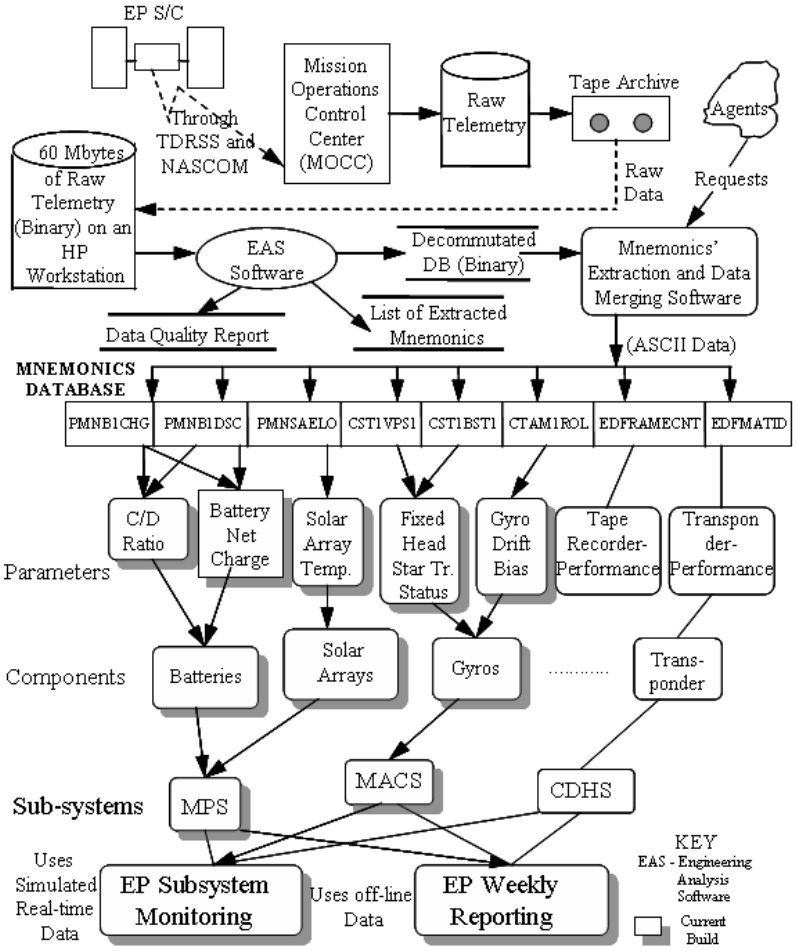

Fig. 4.2. A taxonomy of Explorer Platform satellite system and data extraction process.

and process and reason with the data to support autonomous operation of the control center. This was achievable with the aid of a data server agent that could interface directly with satellite telemetry and provide the information as mnemonics to other specialist agents. Due to operational restrictions, magnetic tapes were used to transfer satellite data to a workstation for processing by a data server agent. Even with that restriction, it was possible to demonstrate the feasibility of an automated agent-based satellite operations control center.

4.1.2 Implementation Architecture for AFLOAT Prototype

An architecture definition for AFLOAT is shown in Figure 4.3. The Multi-agent System (MAS) employed direct communication between agents without a mediator. All external requests, from either a user or a remote client, entered the AFLOAT system through the Interface Services Agent (ISA), which then forwarded them to appropriate agents. The Systems Services Agent (SSA) maintained a database of agents' names, skills, and location (i.e., TCP/IP socket address). It also monitored the health and status of each agent, and provided essential resources for agent migration.

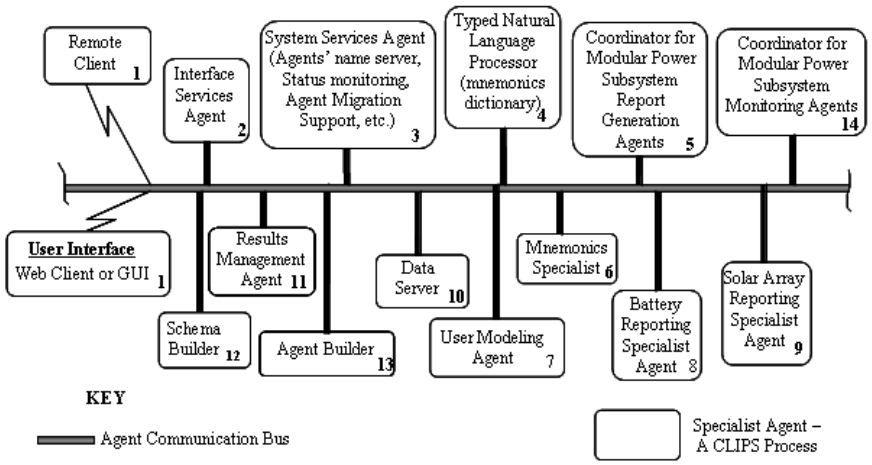

Fig. 4.3. AFLOAT architecture definition.

The architecture provided two classes of specialist agents—coordinator specialist and regular specialist. The coordinator specialist handled complex requests requiring the participation of three or more regular specialist agents to process. The coordinator agent had access to the system's global skill-base and possessed the capability to assemble a group of specialist agents, decompose the request into smaller tasks, and delegate the requests to them. Upon completing their tasks, the specialist agents returned their results to the coordinator, which in turn assembled them and freed up members of the agent group to return to their original states. Each specialist agent also had the capability to collaborate with other agents to process a task.

After accepting a request, a specialist agent examined it to determine whether it would require the services of other agents. If another agent's skill was required to support the task, it sent a message to the SSA requesting the location of the agent. After receiving a response from SSA, it formulated a request and sent a message to the other specialist agent. The other specialist agent would process the task and return a response (result) to the specialist

agent. Each agent depicted in Figure 4.3 existed in a C-Language Integrated Production System (CLIPS) [46] process with a persistent socket connection to the SSA for monitoring its health and safety.

Approaches For Addressing Multi-Agent Architectural Issues In AFLOAT

To successfully develop AFLOAT as a multi-agent system, four architectural issues had to be addressed.

- An approach was established for describing and decomposing the tasks that gave the coordinator specialist agents and regular specialist agents the capability to describe and decompose tasks.
- A format was defined for interaction and communication between agents that employed a semi-structured message format for defining a language protocol for the agents.
- A strategy was formulated for distributing controls among agents. The control strategy initiated by either the coordinator agent or the regular specialist agent was driven by the requirements of the request or the task at hand. In certain situations, control was distributed among the agents, and in other situations, the coordinator agent assigned all tasks to a group of specialist agents in the form of a semi-centralized control framework.
- A policy for coordinating the activities of agents was employed. Our design allowed the SSA to maintain a directory of the skills of all the agents and dispense information on the location of other agents upon request. In addition, SSA monitored the status of each agent and would reactivate, clone, or migrate them when necessary.

4.1.3 The Human Computer Interface in AFLOAT

A major component of AFLOAT was the User Interface Agent (UIA). The UIA was ultimately responsible for supporting dialogs and interactions with outside users of the agent system. In reality, the UIA was a community of agents each with specific tasks. The UIA had a user agent that supported multimodal interaction techniques. As an example, the user could communicate with AFLOAT via typed text or spoken language. There was a Request Analysis Agent that checked for ambiguities in the user's request, checked spelling, filtered superfluous words, and performed pattern recognition and context-dependent analysis. A major component of the UIA was the User Modeling Agent. This agent was responsible for developing user profiles, classifying users, dynamically adapting to user behaviors and preferences, and resolving ambiguities. The Results Management Agent was responsible for interaction with the community of domain specialist agents, collection of results of work done by the domain specialist agents, integration of results, and notification of users. The UIA also supported a local request server that provided

the UIA user-environment management, common services such as e-mail and printing, and results-display support.

AFLOAT also provided an operational domain-restricted natural language interface. The domain restriction is a requirement both for keeping the problem tractable and for performance reasons. This interface allowed users simply to make requests in sentence form. For the grammar component of the natural language interface, a "semantic" grammar was used. This can be defined as a grammar in which the syntax and semantics are collapsed into a single uniform framework [5]. This grammar looks like a context-free grammar except that it uses semantic categories for terminal symbols. There are several benefits of using a semantic grammar, the main one being that there is no need for separate processing of semantics. Semantic processing is done in parallel with syntactic processing. This also means that this method is very efficient, since no time is spent on a separate processing step. There is also the benefit of simplicity. It is not much harder to build a semantic grammar than a syntactical one. All requests either in the form of a natural language or structured queries were submitted to AFLOAT through one of the two interfaces labeled 1 in Figure 4.3. All requests were automatically converted to an Agent Communication Language (ACL) to enable inter-agent communication and collaboration.

4.1.4 Inter-agent Communications in AFLOAT

Inter-agent communication in AFLOAT was based on the following assumptions:

1. Agents in AFLOAT communicated through an asynchronous message passing mechanism (i.e., asynchronous input and output messages),
2. A common message structure was maintained for all agents, and
3. Communication between agents was achieved through TCP/IP-based sockets.

The format of the ACL employed in AFLOAT was an enhanced version of that proposed by Steve Laufmann [104] for "coarse-grained" agents, to which we have added "performatives" proposed by Finin and group for the Knowledge Query Manipulation Language (KQML) [24, 82], and enhanced to meet specific requirements of the domain of spacecraft mission operations. The format for an ACL message in AFLOAT was as follows:

msg-id – A unique identifier composed of hostname, a random number, and system time separated by dashes (e.g., kong-gen854-13412.35).
user-id – The user-id of the person who originally submitted the request.
sender – The name of the agent that the message is coming from.
receiver – The name of the agent that the message is going to (in general). Asterisk (*) is used when we want the agent that is receiving the message to decide to whom the message should finally go.

respond-to – The name of the agent to which the response to this request should go. It does not apply to response messages.
reply-constraint – This is used for time constraints (e.g., ASAP, soon, whenever)
language – The programming language that the message expects. This is especially important when the performative is "evaluate" and the string passed in the *input-string* slot is just evaluated. This would only apply to interpreted languages (e.g., CLIPS, shell scripts, etc.).
msg-type – The message type (request, response, status, etc.).
performative – The basic task to be performed (e.g., parse, archive, generate).
recipient – The name of the user interface agent of the person to receive the results of the request.
result-action – The type of action to invoke on the result message (display, print, etc.).
domain – The domain is the system the message is dealing with. The domain in this case is usually "EP" (Explorer Platform).
object-type – The type of the main object that the message refers to. This is a level in the object hierarchy (e.g., system, subsystem, or component).
object-name – The actual name of the object given in the user request.
object-specifiers – Any words from the user request that give more detailed information about which object is being referred to. This covers specification of a number of objects that are being referred to (e.g., "number 1", "this", "any").
parameters – Any information that adds detail about the task to be performed. For example, when monitoring or reporting on the solar arrays, this field would specify temperature, performance, etc.
action-start – The date/time at which the action being referred to in the message is to begin.
action-duration – The length of time the action is to last.
function – An actual function that is to be called as a result of the message. If the *input-string* field is not empty, it contains parameters that are to be passed to the function.
input-string – This slot contains any data or text information required by other slots of the message. If the function slot is populated, this slot would contain input parameters. For a "parse" performative message, this slot would contain the actual sentence submitted by the user.

Each ACL message format had a message header and a message body. The message header consisted of the message attributes from *msg-id* through *result-format*. The rest of the message was the body of the message. An example of an ACL message generated from a user's natural language input is shown below. A user's input is the value stored in the *input-string* at the bottom of the message. Only pertinent values of the attributes of the message need to be included. Because the performative for this message is "parse",

the system services agent would use the information in its skill base to route the request to the natural language parser for processing. An example of a REQUEST-MESSAGE in an ACL format is the following:

```
(msg-id, gen001)
(user-id, john)
(sender, umbc-ui)
(receiver, loral-coord)
(respond-to, umbc-ui)
(reply-constraint, ASAP)
(language, CLIPS)
 (msg-type, request)
(performative, parse)
(recipient, john-uia)
(result-action, nil)
(result-format, nil)
(domain, "EUVE spacecraft")
 (object-type, nil)
(object-name, nil)
(object-specifiers, nil)
(parameters, nil)
(action-start, nil)
(action-duration, nil)
(function, nil)
(input-string, "monitor the health and safety of the
spacecraft's batteries")
```

Start-up and Activation of a Community of Domain Specialist Agents in AFLOAT

To initiate the prototype, a UNIX process loaded the System Services Agent (SSA), labeled as box 3 in Figure 4.3. The SSA then loaded each of the agents depicted in boxes 2, 4, 5, 6, 7, 8, 9, 10, and 11 and established a persistent socket connection with each of them. The links were persistent to enable the SSA to monitor the status of the nine agents. In addition to monitoring the status of the agents, the SSA stored the location and skills base of other agents and provided appropriate resources to support the agents' migration requirements. To support fault tolerance, the Interface Services Agent (ISA) had the capability to monitor the status of the SSA, and to restart it if it died. Communications from clients (i.e., external interfaces—either user interfaces or remote clients), needed to be registered at the ISA. This was necessary to relieve the processing load of the SSA. The agents numbered 2 and higher constituted the AFLOAT server. All the agents communicated when necessary via TCP/IP sockets. At run time, the User Interface (UIA) was loaded. Users submitted requests to AFLOAT from a remote web client and received responses/results locally.

4.2 Lights Out Ground Operations System

LOGOS, the Lights Out Ground Operations System [175, 177, 181, 183], was a proof-of-concept system that used a community of autonomous software agents that worked cooperatively to perform the functions previously undertaken by human operators who were using traditional software tools, such as orbit generators and command-sequence planners. The following discusses the LOGOS architecture and gives an example scenario to show the data flow and flow of control.

4.2.1 The LOGOS Architecture

For reference, an architecture of LOGOS is shown in Figure 4.4. LOGOS was made up of ten agents, some of which interfaced with legacy software, some of which performed services for the other agents in the community, and others of which interfaced with an analyst or operator. All agents could communicate with any other agent in the community, though not all of the agents were required to communicate with other agents.

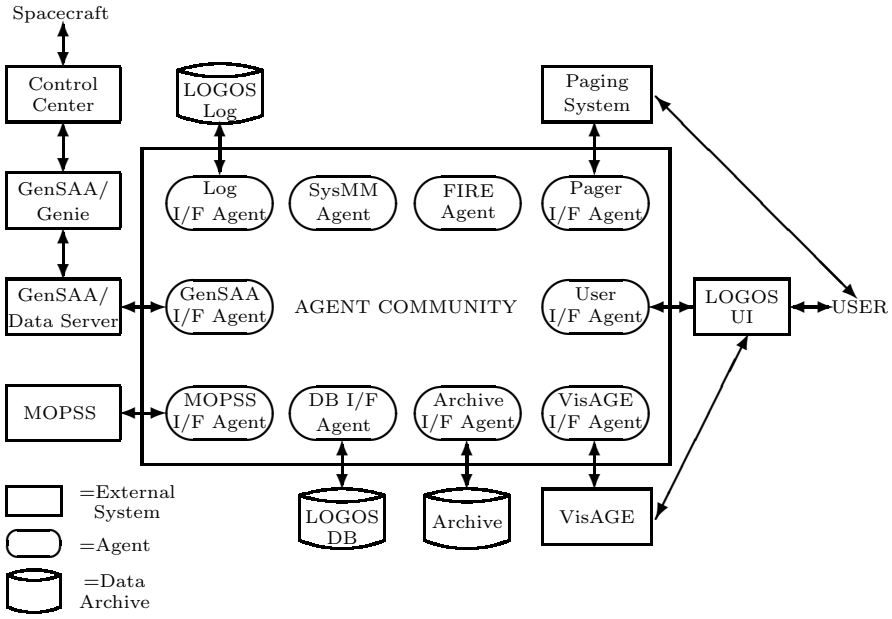

Fig. 4.4. LOGOS agent architecture.

The System Monitoring and Management Agent (SysMMA), kept track of all of the agents in the community and provided addresses of agents for other agents requesting services, similar to the Interface Services Agent in

AFLOAT. Each agent when started had to register with SysMMA to register their capabilities and to obtain addresses of other agents whose services it needed.

The Fault Isolation and Resolution Expert (FIRE) agent resolved satellite anomalies. FIRE was notified of anomalies during a satellite pass. It contained a knowledge base of potential anomalies and a set of possible fixes for them. If it did not recognize an anomaly or was unable to resolve it, it then sent the anomaly to the user interface agent to be forwarded to a human analyst for resolution.

The User Interface Agent (UIFA) was the interface between the agent community and the graphical user interface that the analyst or operator used to interact with the LOGOS agent community. UIFA received notification of anomalies from the FIRE agent, handled login of users to the system, kept the user informed with reports, routed commands to be sent to the satellite, and performed other maintenance functions. If the attention of an analyst was needed but none was logged on, UIFA would send a request to the PAGER agent to page the required analyst.

The VisAGE Interface Agent (VIFA) interfaced with the VisAGE (Visual Analysis Graphical Environment) data visualization system. VisAGE was used to display spacecraft telemetry and agent log information. Real time telemetry information was displayed by VisAGE as it was downloaded during a satellite pass. VIFA requested the data from the GIFA and AIFA agents (see below). An analyst could also use VisAGE to visualize historical information to help monitor spacecraft health or to determine solutions to anomalies or other potential spacecraft problems.

The Pager Interface Agent (PAGER) was the agent community interface to the analyst's pager system. If an anomaly occured or other situation arose that needed an analyst's attention, a request was sent to the PAGER agent, which then paged the analyst.

The Database Interface Agent (DBIFA) and the Archive Interface Agent (AIFA) stored short term and long term data, respectively, and the Log agent (LOG) stored agent logging data for debugging and monitoring purposes. The DBIFA stored information such as a list of the valid users and their passwords, and the AIFA stored telemetry data.

The GenSAA/Genie Interface Agent (GIFA) interfaced with the GenSAA/Genie ground station software [65], which handled communications with the spacecraft. GenSAA/Genie was used to download telemetry data and maintain scheduling information, and was used to upload commands to the spacecraft. As anomalies and other data were downloaded from the spacecraft, GIFA routed the data to other agents based on their requests for information.

The MOPSS (Mission Operations Planning and Scheduling System) Interface Agent (MIFA) interfaced with the MOPSS ground-station planning and scheduling software. MOPSS kept track of the satellite's orbit and when the next pass would occur and how long it would last. It also sent out updates to the satellite's schedule to requesting agents when the schedule changed.

4.2.2 An Example Scenario

An example scenario illustrating how the agents would communicate and cooperate would start with MIFA receiving data from the MOPSS scheduling software informing MIFA that the spacecraft would be in contact position in two minutes. MIFA would then send a message to the other agents to wake them up, if they were sleeping, and let them know of the upcoming event. The advance notice allowed them to do some preprocessing before the contact. When GIFA received the message from MIFA, it would send a message to the GenSSA Data Server to put it into the proper state to receive transmissions from the control center.

After receiving data, the GenSSA Data Server would send the satellite data to GIFA. GIFA had a set of rules that indicated which data to send to which agents. As well as sending data to other agents, GIFA also sent all engineering data to the archive agent (AIFA) for storage, and sent trend information to the visualization agent (VIFA). Updated schedule information was sent to the scheduling agent (MIFA) and a report was sent to the user interface agent (UIFA) to send on to an analyst for monitoring purposes. If there were any anomalies, they were sent to the FIRE agent for resolution.

If there was an anomaly, the FIRE agent would try to fix it automatically by using a knowledge base containing possible anomalies and a set of possible resolutions for each anomaly. To fix an anomaly, FIRE would send a spacecraft command to GIFA to be forwarded on to the spacecraft. After exhausting its knowledge base, if FIRE was not able to fix the anomaly, FIRE would forward the anomaly to the user interface agent, which then paged an analyst and displayed the anomaly on the analyst's computer for action. The analyst would then formulate a set of commands to send to the spacecraft to resolve the situation. The commands would then be sent to the FIRE agent so it could add the new resolution to its knowledge base for future reference. The commands then would be sent to the GIFA agent which sent them to the GenSAA/Genie system for forwarding on to the spacecraft.

There were many other interactions between the agents and the legacy software that were not covered above. Examples include the DBIFA requesting user logon information from the database, the AIFA requesting archived telemetry information from the archive database to be sent to the visualization agent, and the pager agent sending paging information to the paging system to alert an analyst of an anomaly needing his or her attention.

4.3 Agent Concept Testbed

The motivation behind ACT (Agent Concept Testbed) was to develop a more flexible architecture than LOGOS for implementation of a wide range of intelligent or reactive agents. After developing the architecture, sample agents were built to simulate ground control of a satellite constellation mission as

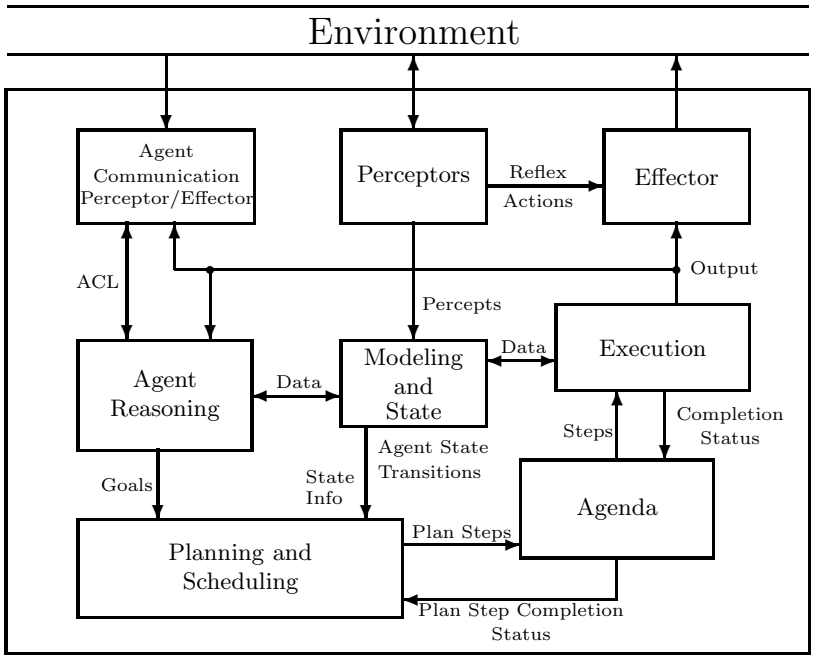

Fig. 4.5. ACT Agent Architecture.

a proof of concept. The following discusses the ACT agent architecture and gives an operational scenario using the satellite constellation proof of concept.

4.3.1 Overview of the ACT Agent Architecture

The ACT architecture was a component-based architecture that allowed greater flexibility to the agent designer. A simple agent could be designed by using a minimum number of components that would receive percepts (inputs) from the environment and react relative to those percepts. This type of simple agent would be a *reactive agent*.

A robust agent could be designed using more complex components that allowed the agent to reason in a deliberative, reflexive, and/or social fashion. This robust agent would maintain models of itself, other agents in its environment, objects in the environment that pertain to its domain of interest, and external resources that it might utilize in accomplishing a goal. Figure 4.5 depicts the components for a robust agent. The depicted components gave the agent a higher degree of intelligence when interacting with its environment.

The ACT agent architecture was capable of several types of behaviors. Basically, "agent behavior" refers to the manner in which an agent responds to some sort of stimulus, generated either externally (outside the agent) or internally (within the agent). We have identified four basic classes of behaviors agents can realize. These are:

Social – Social behaviors refer to behaviors shared between/among agents. The ACT architecture supported two types of social behavior: social behavior triggered by another agent and social behavior triggered by the agent itself. In each of these cases, the agent utilized ACL messages to solicit help or to coordinate the behaviors of other agents.

Proactive – This type of behavior is stimulated in some way by the agent itself. For our agents, there was one type of proactive behavior that was be supported: self motivating. Self-motivating behaviors are triggered by built-in or intrinsic goals.

Reactive – Reactive behaviors are those that require "no thinking". These behaviors are like built-in reflexive actions that are triggered by events in the agent's environment or by other agents. When detected, the agent responds immediately with a predetermined action.

Deliberative – This type of behavior is perhaps the most difficult and interesting. At the highest level of abstraction, this type of behavior involves the establishing of a hierarchy of goals and subgoals, the development of plans to achieve the subgoals, and the execution of the planned steps to ultimately accomplish the goal that started the process of deliberation in the first place.

4.3.2 Architecture Components

Components

A component in the agent architecture is a software module that performs a defined task. Components when combined with other software components can constitute a more robust piece of software that is easily maintained and upgraded. Each component in the architecture can communicate information to/from all other components as needed through various mechanisms including a publish-and-subscribe communication mechanism, message passing, or a request for immediate data.

Components may be implemented with a degree of intelligence through the addition of reasoning and learning functions. Each component needs to implement certain interfaces and contain certain properties. Components must implement functionality to publish information, subscribe to information, and accept queries for information from other components or external resources being used by the component. Components need to keep track of their state, and need to know what types of information they contain and what they need from external components and objects.

The following describes the components in the ACT agent architecture.

Modeler

The modeling component was responsible for maintaining the domain model of an agent, which included models of the environment, other agents in the

community, and the agent itself. The Modeler received data from the Perceptors and agent communication component. This data was used to update state information in its model. If the data caused a change to a state variable the Modeler then published this information to other components in the agent that subscribed to updates to that state variable. The Modeler was also responsible for reasoning with the models to act proactively and reactively with the environment and events that affected the model's state.

The modeler could also handle what-if questions. These questions would primarily originate from the planning and scheduling component, but could also come from other agents or from a person who wanted to know what the agent would do in a given situation or how a change to its environment would effect the values in its model.

Reasoner

The Agent Reasoner made decisions and formulated goals for the agent component through reasoning with received ACL messages and information in its local knowledge base, as well as with model and state information from the Modeler. This component reasoned with state and model data to determine whether any actions needed to be performed by the agent to affect its environment, change its state, perform housekeeping tasks, or perform other general activities. The Reasoner would also interpret and reason with agent-to-agent messages received by the agent's communications component. When action was necessary for the agent, the Reasoner would produce goals for the agent to achieve.

Planner/Scheduler

The Planner/Scheduler component was responsible for any agent-level planning and scheduling. The Planning component formulated a plan for the agent to achieve the desired goals. The planning component was given a goal or set of goals to fulfill in the form of a plan request. This typically came from the Reasoner component but could be generated by any component in the system.

At the time that the plan request was given, the planning and scheduling component acquired a state of the agent and system, usually the current state, as well as the set of actions that could be performed by this agent. This information would typically be acquired from the modeling and state component. The planning and scheduling component then generated a plan as a directed graph of steps. A step is composed of preconditions to check, the action to perform, and the expected results from the action (post condition). When each step was created, it was passed to Domain Expert components/objects for verification of correctness. If a step was deemed incorrect or dangerous, the Domain Expert could provide an alternative step, solution, or data to be considered by the planner.

Once the plan was completed, it was passed back to the component that requested the plan (usually the Reasoner). The requesting component then either passed it on to the Agenda to be executed or used it for planning/what-if purposes.

Agenda/Executive

The Execution component managed the execution of steps and determined the success or failure of each step's execution. Output produced during a step's execution could be passed to an Effector or the Reasoning component. The Agenda and Executive worked together to execute the plans developed by the Planner/Scheduler. The agenda typically received a plan from the Reasoner, though it could receive a plan from another component that was acting in a reactive mode. The agenda interacted with the Execution component to send the plan's steps in order for execution. The agenda kept track of which steps were being executed, had finished executing, were idle, or were waiting for execution. It updated the status of each step appropriately as the step moved through the execution cycle. The agenda reported the plan's final completion status to the Planner and Agent Reasoner when the plan was complete.

The Executive would execute the steps it received from the Agenda. A step contained preconditions, an action, and possible post-conditions. If the preconditions were met, the action was executed. When executions finished, the post-conditions were evaluated, and a completion status was generated for that step. The completion status was returned to the agenda, which allowed for overall plan evaluation.

The execution component interacted with the agenda in the following way. The agenda sent the first step to the execution component. This woke up the Executive. The component then began executing that step. The Executive then checked to see if another step was ready for execution. If not, the component would go back to sleep until it received another step from the agenda.

The Modeling component would record state changes caused by a step execution. When a plan was finished executing, the Agenda component sent a completion status to the Reasoning component to indicate that the goal established by the Reasoner had been accomplished. If the Agent Reasoning component was dealing with data from the environment, it could decide either to set a goal (for more deliberative planning) or to react quickly in an emergency situation. The Reasoner could also carry on a dialog with another agent in the community through the Agent Communication Perceptor/Effector.

A watch was also attached to the Executive. It monitored given conditions during execution of a set of steps and the consequence if the condition occurred. Watches allowed the planner to flag things that had to be particularly looked out for during real-time execution. They could be used to provide "interrupt" capabilities within the plan. An example of a watch might be to monitor drift from a guide star during an observation. If the drift exceeds a threshold, the observation is halted. In such a case, the watch would notify the

Executive, which in turn would notify the Agenda. The Agenda would then inform the Reasoner that the plan had failed and the goal was not achieved. The Reasoner would then formulate another goal (e.g., recalibrate the star tracker).

Agent Communications

The agent communication component was responsible for sending and receiving messages to/from other agents. The component took an agent data object that needed to be transmitted to another agent and converted it to a message format understood by the receiving agent. The message format that was used was based on FIPA [110]. The message was then transmitted to the appropriate agent through the use of a NASA-developed agent messaging protocol/software called Workplace [7].

The reverse process was performed for an incoming message. The communications component took the message and converted it to an internal agent object and sent it out to the other components that had a subscription to incoming agent messages. The communications component could also have reactive behavior where for a limited number of circumstances it could produce an immediate response to a message.

Perceptors/Effectors

Percepts received through sensors, communication with external software/systems, and other environmental entities were received through a Perceptor component. These percepts were passed from the Perceptor to the Modeling component, where a model's state was updated as needed.

The Perceptors were responsible for monitoring parts of the environment for the agent. An example might be a subsystem of a spacecraft or recurring input from a user. Other than agent-to-agent messages, any data received by the agent from the environment entered through Perceptors. An agent might have zero or more Perceptors, where each Perceptor received information from specific parts of the agent's environment. A Perceptor could just receive data and pass it on to another component in the agent, or it might perform some simple filtering/conversion before passing it on. A Perceptor might also act intelligently through the use of reasoning systems. If an agent was not monitoring a part of the environment, then it would not have any perceptors (an example of this would be an agent that only provides expertise to other agents in a certain area, such as fault resolution).

The Effector was responsible for effecting or sending output to the agent's environment. Any agent output data, other than agent-to-agent messages, left through Effectors. Typically the data coming from the Effectors would be sent from the executive that had just executed a command to the agent's environment. There could be zero or more Effectors, where each Effector sent data to specific parts of the agent's environment. An Effector could perform data

conversions when necessary and could even act intelligently and in a proactive manner when necessary through the use of internal reasoning systems. As with the Perceptors, an agent might not have an Effector if it did not need the capability of interacting with the environment.

Agent Framework

A software *framework*, into which the components were "plugged", provided a base functionality for the components as well as the inter-component communication functionality. The framework allowed components to easily be added and removed from the agent while providing for a standard communications interface and functionality across all components. This made developing and adding new components easier and made component addition transparent to existing components in the agent.

The communications mechanism for components was based on a publish-and-subscribe model, with a direct link between components when there was a large amount of data to be transferred. Components communicated to each other the types of data that they produced when queried. When one component needed to be informed of new or changed data in another component, it identified the data of interest and subscribed to it in the source component. Data could be subscribed-to whenever they are changed or on an as-needed basis. With this mechanism, a component could be added or removed without having to modify the other components in the agent.

4.3.3 Dataflow Between Components

This section gives an example of how data flowed between components of the architecture. In this example scenario, a spacecraft's battery is discharging. Figure 4.6 shows a timeline and the flow of data between components. The following is the scenario:

1. The agent detects a low voltage by reading data from the battery via a Perceptor. The Perceptor then passes the voltage value to the Modeler, which has subscribed to the Perceptor to receive all percepts.
2. When the Modeler receives the voltage from the Perceptor, it converts the voltage to a discrete value and updates this value in the model. In this case, the updated voltage value puts it below the acceptable threshold and changes the model's voltage state to "low". This change in state value causes a state change event and the Modeler now publishes the new state value to all components that have subscribed to changes in this state variable. Since the Reasoner has subscribed to changes in this state variable, the low voltage value is sent to the Reasoner.
3. In the Reasoner, the low voltage value fires a rule in the expert system. This rule calls a method that sends the Planner/Scheduler a goal to achieve a battery voltage level that corresponds to fully charged.

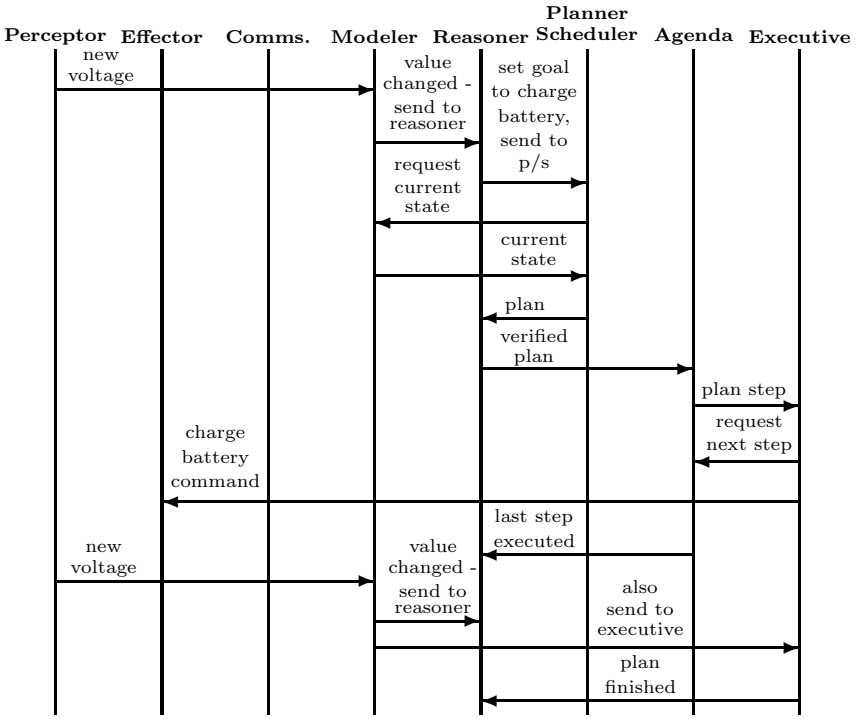

Fig. 4.6. Scenario of data flowing between agent components.

4. When the Planner/Scheduler receives the goal from the Reasoner, it queries the Modeler for the current state of the satellite and a set of actions that can be performed (this set may change based on the health of the satellite).
5. After receiving the current state of the satellite and the set of available actions from the Modeler, the Planner/Scheduler formulates a list of actions that need to take place to charge the battery. It then sends the plan back to the Reasoner for validation.
6. The Reasoner examines the set of actions received from the Planner/Scheduler and decides that it is reasonable. The plans are then sent to the Agenda.
7. The Agenda then puts the action steps from the plan into a queue for the Executive.
8. As the Executive is ready to execute a new step, the agenda passes the plan steps one at a time to the Executive for execution.
9. The Executive executes each action until the plan is finished. At this time the Executive notifies the Agenda that it has finished executing the plan.
10. The Agenda marks the plan as finished and notifies the Reasoner (or whomever sent the plan) that the plan finished successfully.

11. After the plan is executed, the voltage starts to rise and will trigger a state change in the Modeler when the voltage goes back into the fully charged state. At this time, the Reasoner is again notified that a change in a state variable has occurred.
12. The Reasoner then notes that the voltage has been restored to the fully charged state and marks the goal as accomplished.

4.3.4 ACT Operational Scenario

The operational scenario that was developed to evaluate ACT was loosely based on certain nanosatellite constellation ideas. Figure 4.7 graphically illustrates this scenario. It was based on the idea of a ground-based community of proxy agents (each representing a spacecraft in the nanosatellite constellation) that provided autonomous operations of the constellation. Another scenario corresponded to the migration of this community of proxy agents to the spacecraft themselves, to support an evaluation of space-based autonomy concepts.

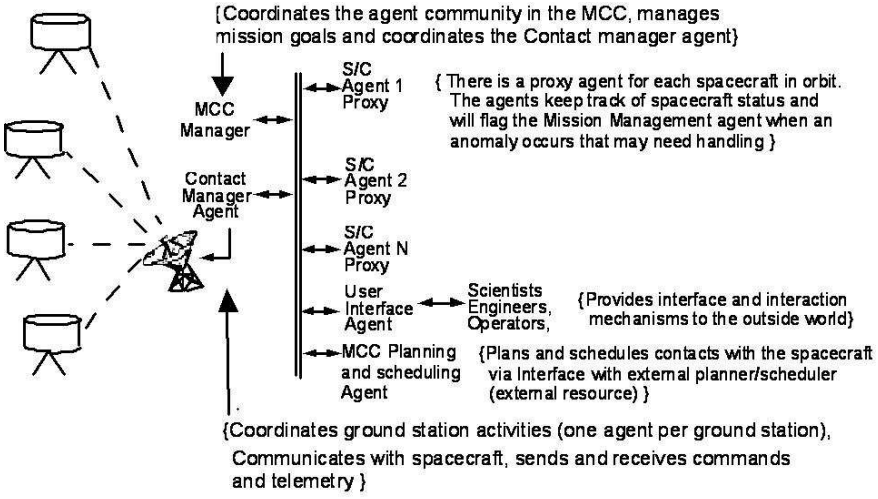

Fig. 4.7. Agent community developed in ACT to test the new agent architecture and community concepts.

In this scenario, several nanosatellites are in orbit collecting magnetosphere data. The Mission Operations Control Center (MOCC) makes contact with selected spacecraft according to its planned schedule when the spacecraft (S/C) come into view.

The agents that would make up the MOCC would be:

- Mission Manager Agent: coordinates the agent community in the MOCC, manages mission goals, and coordinates Contact Manager Agents.
- Contact Manager Agent: coordinates ground station activities (one agent per ground station), communicates with the spacecraft, and sends and receives data, commands, and telemetry.
- User Interface: interfaces with the user to accept commands for the spacecraft and sends data to be displayed.
- MOCC Planning/Scheduling Agent: plans and schedules contacts with the spacecraft via interface with external planner/scheduler.
- Spacecraft Proxy Agents: there is one proxy agent for each spacecraft in orbit. The agents keep track of spacecraft status, health and safety, etc. The agents will notify the Mission Manager Agent when there occurs an anomaly that may need handling.

Each of the above agents registers with the GCC manager agent. The GCC manager agent notifies the agents when there is an impending contact for their spacecraft, and when another agent is going to be added to the community; it also provides to the agents the address of the other agents (so agents can pass messages to each other). The following is a spacecraft contact scenario that illustrates how the agents worked with the GCC manager agent:

- Agents register with the GCC Manager Agent at system startup.
- The GCC Planner/Scheduler Agent communicates with the spacecraft Proxy Agents to obtain spacecraft communications-view data. It then creates a contact schedule for all orbiting spacecraft.
- The GCC Manager Agent receives the schedule from the GCC Planner/Scheduler Agent.
- The GCC Manager Agent informs the Contact Manager Agent about the next contact (when and with which spacecraft).
- The Contact Manager Agent (CMA) receives notification of an acquisition of signal (AOS) from a spacecraft. The MOCC is now in contact with the spacecraft.
- The CMA executes the contact schedule to download data, delete data, or save data for a future pass.
- The Contact Manager Agent analyzes the downloaded telemetry data. If the telemetry indicates a problem, the CMA may alter the current contact schedule to deal with the problem.
- The CMA performs any necessary commanding in parallel with any data downloads.
- The Contact Manager Agent sends the telemetry to the appropriate spacecraft Proxy Agent for processing.
- The spacecraft Proxy Agent processes the telemetry data and updates the state of its model of the spacecraft from the telemetry received.
- If the spacecraft Proxy determines that a problem exists with the spacecraft and an extended or extra contact is needed, a message is sent to the

GCC Planner/Scheduler Agent which will re-plan its contact schedule and redistribute it to the GCC Manager.
- The spacecraft Proxy Agent sends to the Contact Manager any commands that need to be uploaded.
- The Mission Manager Agent ends contact when scheduled.

4.3.5 Verification & Correctness

Whereas AFLOAT and LOGOS demonstrated that typical control center activities could be emulated by a multi-agent system, the major objective of the ACT project was to demonstrate that ground-based surrogate agents, each representing a spacecraft in a group of spacecraft, could control the overall dynamic behaviors of the group of spacecraft in the realization of some global objective. The ultimate objective of ACT was to help in the understanding of the idea of progressive autonomy (see section 9.6) which would, as a final goal, allow the surrogate agents to migrate to their respective spacecraft and then allow the group of autonomous spacecraft to have control of their dynamic behaviors independent of relying on ground control.

ACT properly emulated the correct interaction between surrogate agents and their respective spacecraft. This "correctness" was determined by comparison between what the surrogate did versus what a human controller on the ground would have done, in conjunction with what the controllers associated with the other surrogates would have done, to achieve a global objective. This analysis was undertaken more at the heuristic level than at a formal level. The design of the surrogates was realized in a modular fashion in order to support the concept of incremental placement of the functional capabilities of the surrogate agent in the respective spacecraft, until the spacecraft itself was truly agent-based and "autonomous". This particular aspect of the ACT project was heuristically realized but not rigorously (formally) tested out.

The use of formal methods has been identified as a means of dealing with this complex problem. Formal approaches were previously used in the specification and verification of the LOGOS system [56, 118, 119, 124]. A formal specification in CSP highlighted a number of errors and omissions in the system. These, and other, errors were also found by an automated tool [57, 58, 59, 111, 112] which implemented an approach to requirements-based-programming [52]. For more information on formal verification of agent-based systems, see [123].

Part II

Technology

5

Core Technologies for Developing Autonomous and Autonomic Systems

This chapter examines the core artificial intelligence technologies that will make autonomous autonomic spacecraft missions possible. Figure 5.1 is a pictorial overview of the technologies that will be discussed. The *plan* technologies will be discussed first followed by the *act* and *perceive* technologies and finally technologies appropriate for testing.

It is difficult to make definitive statements on the functionality, strengths, and weaknesses of software systems in general, since designers have tremendous latitude on what they do. This chapter explains and discusses the attributes seen in a majority of the systems described. It should be understood that exceptions may exist.

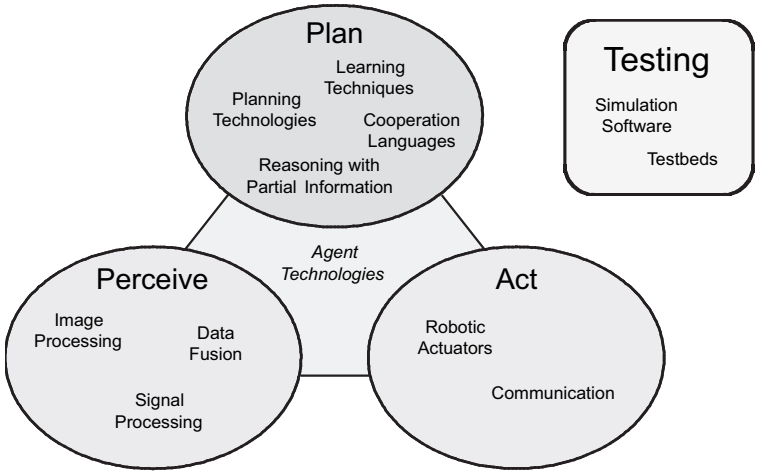

Fig. 5.1. Collaborative Autonomy Technologies

5.1 Plan Technologies

The planning portion of the autonomy cycle is responsible for examining the environment and choosing appropriate actions in light of the goals and mission of the system. Sometimes this choice requires interactions with other systems. Planners are a central technology in all computerized planning, and many techniques have been developed to support planning, such as formal collaboration languages, evidential reasoning, and learning techniques. The rest of this section will discuss these techniques and planning in general.

5.1.1 Planners

A defining characteristic of an autonomous system is the ability to independently select appropriate actions to achieve desired objectives. Planner systems are the software component commonly used to achieve this capability. Work on software planners goes back to 1959 and, over the intervening years, many types of planners have been developed.

Figure 5.2 shows a high level view of a planner and its context in a system architecture. All planners begin with a set of initial mission objectives that are specified as goals to the planner. These goals are analyzed in the light of the planner's view of the environment and a database describing what it is capable of doing. After analysis, the planner chooses a set of actions to perform and these actions are sent off for execution. Results from the execution of these actions are fed back to the planner to update its view of the environment. If the goals have been achieved, success is reported back to the higher level system. If some problem has occurred during execution, error recovery is attempted, a new plan is created, and the cycle repeats. If no other plan can be created, the failure is reported to the higher level system.

This description is generic and it leaves many design decisions unanswered. For example, planners differ in how they describe the goals they are to achieve, the environment they execute in, and their database of potential actions. Each of these descriptions is given in a computer-based language. This language is very important since it defines what the system is capable of doing and how it will do it. It also defines what the system cannot do. If the language used is too limited, it may not be able to describe some aspect of the domain, potentially limiting the planner's ability to handle some situations.

Planners differ in the speed at which they come to decisions. Some planners are slow and deliberative while others are quick and reactive. In general, the slow deliberative planners make plans that are more globally optimal and strategic in nature. The reactive planners tend to examine the environment and choose from a highly constrained set of plans. They are tactical in nature and work well in rapidly changing environments where the time for slow careful choice is not available. In many real world systems, either the planner is designed to handle both deliberation and reactivity, or two separate planners are integrated together.

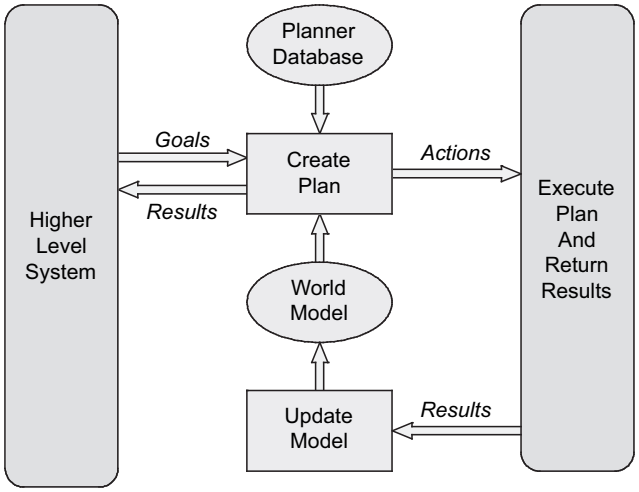

Fig. 5.2. Planner Architectures

All robust planners must deal with the failure of an action during execution. Some planners have low level strategies on hand and when a failure occurs, they immediately attempt to repair the plan. Others have a database of alternative ways to achieve an objective, and when an attempt fails, they analyze the current environment and choose another plan. In domains where the environment changes very rapidly relative to the planner decision time or where action failure is a regular occurrence, the planner may take the possibility of failure into consideration during plan creation. These systems give preference to robust plans that can recover from likely failures even if the robust plan has a higher cost in resources than the alternatives.

Some planners convert higher level tasks into low level actions just before execution. By using this strategy, they commit fewer resources to any one plan and can quickly react to changes in the environment. Unfortunately, the plan that is ultimately executed will often be sub-optimal, particularly if two or more tasks are competing for the same resource. Other planners map the top-level goals into a complete series of small actions that take place in a time-sequenced manner. The advantage of this approach is that a more globally optimal plan can be created. Its disadvantage is that a failure occurring in one step can cause the rest of the plan to be abandoned and re-planned. This is computationally expensive and time consuming. Re-planning can also cause problems if the domain looks forward into the plan and begins the commitment of resources based on the expected plan. In these domains, the re-planning step must use repair strategies in an attempt to maintain most of the original plan.

Planners must be sensitive to an action's cost in resources. In computer domains, such as software agents, actions have small costs and plan selection can usually ignore resource issues. In other domains, like spacecraft, some actions commit resources that cannot be replenished (such as propellant). When resources have a very high cost, a failure during an action can threaten the whole mission. For planning in this type of domain, the costs and recovery strategies must be carefully chosen before action is taken and resources committed.

Having reviewed planner technology and a number of design choices facing a planner, we now will describe several common planner technologies.

Symbolic Planners

Symbolic planners are systems that represent their goals and plans as a series of symbolic assertions instead of numbers, fuzzy quantities, or probabilities. Symbolic planners have been used in many domains.

Figure 5.3 depicts a symbolic planner. The plans of a symbolic planner are stored in a centralized database. Each plan has a set of preconditions and a list of operations to perform. The planner uses the preconditions to determine when it can use a plan. These preconditions can specify environmental constraints, the availability of resources, and potentially whether other plans have been executed before this plan. The list of operations in the plan may be primitive operations to perform or they can be additional plan components. These plan components become additional sub-goals that need to be examined by the planner. Often a plan instantiated during the planning cycle is called a *task*.

While symbolic planners vary in detail, they generally start the planning activity by examining the goals in light of the current environment and then break the goals into a series of tasks. These tasks are themselves examined and broken into simpler subtasks. This iterative refinement process continues until a point is reached where the tasks define a series of steps at an appropriate level of abstraction for plan implementation. The plan is executed and feedback is generated on the success of the tasks. The higher level system is signaled when objectives have been met. If a failure occurs in one or more of the tasks, the planner modifies its plan and the cycle is repeated. If the planner exhausts all of its options and the objectives have not been meet, the planner signals to the higher level that it has failed.

Symbolic planners use many different strategies for choosing among the potential plans. They often spend large amounts of computer resources generating plans and selecting the best ones. They have difficulties in situations where the decision cycle time is short or where actions and failures are not deterministic.

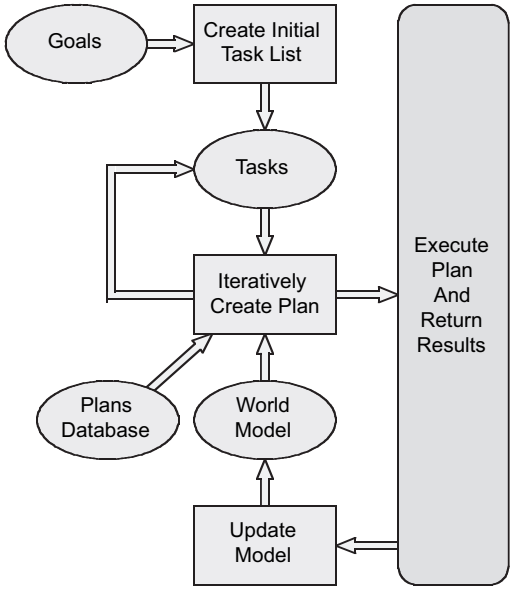

Fig. 5.3. Symbolic Planner

Reactive Planners

Reactive planners are specifically designed to make rapid choices in time critical situations. They can be designed around either symbolic or numeric representations.

Figure 5.4 shows the structure of a reactive planner. Reactive planners begin by evaluating the available plans in light of the current context and then choosing the most appropriate. These plans are simple in nature and are designed to execute immediately. Once the plan is selected and executing, the planner monitors the current situation and only changes the plan if the goal is modified or the plan succeeds or fails. A reactive planner can quickly assess the updated situation and switch to a new plan. The selection process is kept fast by limiting the number of potential plans that have to be examined. Sometimes this is accomplished by the higher-level control system explicitly listing all plans the reactive planner needs to examine. On other systems, the set of plans is automatically constrained by indexing the plans on their applicable context. At each decision step, only the context-appropriate plans will have to be examined.

Reactive planners spend little time choosing the next plan and therefore are appropriate for time critical situations. They are often used in the low-level control of robot and robot-like systems, which have hard real-time deadlines. Because reactive planners only examine a limited number of choices, they will

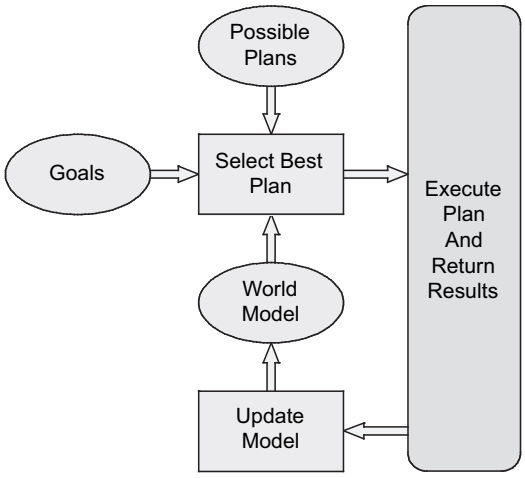

Fig. 5.4. Reactive Planner

often make good local decisions and poor global ones. Another component of the overall system architecture is usually made responsible for optimizing the global objectives.

Model-Based Planners

Model-based planners use models to analyze the current situation and create their plans. They represent this information symbolically with goals described as desired changes in the environment.

Figure 5.5 shows the structure of a model-based planner. These systems give careful focus to the process of updating their internal models. As part of world-model update, these planners may create detailed sub-models of the internal status of sensors and actuators, and, using their model, they can extend the direct measurements to determine the status of internal, but not directly measured, subsystems.

Once an accurate model of the platform and world has been created, the model-based planner examines the goals, and, using its models, creates a plan to achieve the objectives. Model-based systems can be extremely effective at creating plans in the face of one or more damaged subsystems.

One can argue that many planners are model-based since their database of potential actions implicitly defines the system and its capabilities. While there is some merit to this argument, it misses the fact that this implicit knowledge is usually generated by human programmers and may not be complete. These systems are therefore artificially limited in what they can plan. A model-based planner, using its deep understanding of the system it controls, can come up with novel approaches to achieve the objectives. This is most important

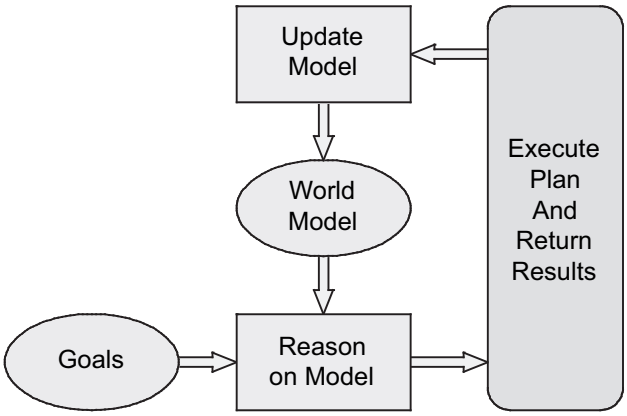

Fig. 5.5. Model-Based Planning

when the system is damaged. Model-based systems can use their model to work around the damage. In other approaches, the human programming must explicitly define the strategy to use when failure occurs, and if they did not make a plan for the failure, the system will respond sub-optimally.

The advantages of model-based approaches are that they only need the model description of the system being controlled and the rest is automatically determined. Their disadvantage is that reasoning on models can be very slow and the necessary models are often hard to construct and possibly incomplete. Model-based approaches are often used in support of other forms of planners.

Case-Based Planners

Case-based planners are systems that represent their planning knowledge as a database of previously solved planning cases. Case-based planners exploit the idea that the best way to solve a new problem is probably very similar to an old strategy that worked. Their cases can be a mixture of symbolic and numeric information. Case-based planners are a class of Case-Based Reasoning (CBR) technology [1, 116, 138].

Figure 5.6 shows the structure of a case-based planner. When a case-based planner is given a new goal, it attempts to find a similar case in its case database. The cases are indexed on the goals being solved and the environmental conditions when they were solved. If similar cases are found, they are extracted, adapted to the current situation, and analyzed to see whether they will achieve the goal being pursued. Often the case will be simulated in the current environment to determine its actual behavior. This simulation can discover defects in the plan so that repair strategies can be applied to customize the plan for the current situation. The new case is simulated and the cycle repeated. Once a plan has met all the necessary requirements, it is

executed. The results of execution are examined, and if it did not achieve the objective, new cases are retrieved and the cycle is repeated. Cases that are successful may be placed in the database for future planning activity. In this way the case-based planner learns and becomes more proficient.

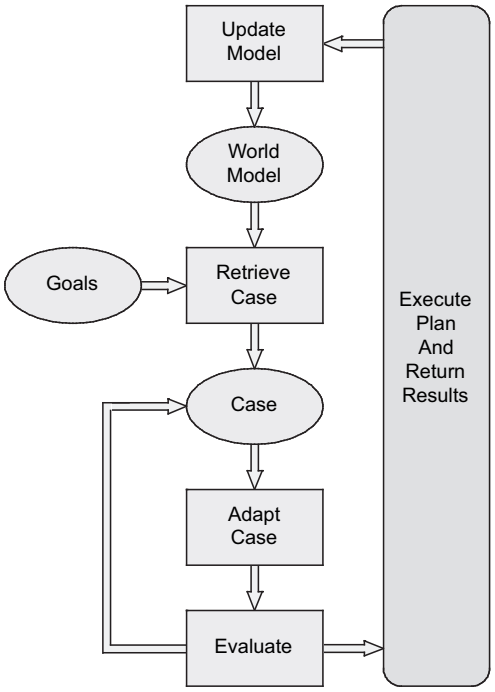

Fig. 5.6. Case-Based Planning

Two difficulties that arise in case-based planners pertain to the methods used to index the cases and the reasoning component of the system. How cases are indexed determines whether appropriate cases will be found when they are needed to solve a problem. If the indexing is too restrictive, an appropriate case may not be retrieved for examination and the knowledge it represents may be lost. The system will have to start the planning with a less optimal case. If the indexing is too loose, a large number of inappropriate cases will be supplied and each will have to be examined and eliminated. This makes the resulting system slower and less responsive. How to index the cases is a central problem in case-based systems and in a real sense can determine its success.

The second difficulty is the reasoning component of a case-based planner. This component is responsible for determining whether the plan will work

and repairing it if the plan can be repaired. Ultimately, if no plan is similar to the current situation, the reasoning component must create a whole new plan. These are difficult components to construct.

Many case-based planners use a conventional symbolic or a model-based planner as the reasoning component of the case-based planner. The advantage to this approach is that the resulting system has all of the capabilities of the conventional symbolic or a model-based planner with a case-based planner's ability to learn new cases. In effect, the case-based database generated is used to cache the knowledge of the conventional system. This makes the resulting system more responsive over time.

Schedulers

Planning systems and scheduling systems work in the same domain, solve very similar problems, and use the same core technologies. The primary difference is that scheduling systems are more focussed on detailed accounting of resources, and they tend to generate complete and detailed plans.

5.2 Collaborative Languages

Most forms of collaboration require the collaborating agents to communicate using a shared language. This language specifies the kinds of information that can be exchanged between the agents and in a real way determines what one agent can and cannot communicate to another. Well designed languages allow the agents to say what is necessary to solve the problem. Poor languages limit communication and support less than optimal solutions.

Computer systems support many kinds of communication. The simplest communication, from a computer science point of view, is an exchange of messages whose content is an internal program data structure. The data structure can be a simple primitive type like a number or string, or the structure can be complex connections of records. This approach to communication is easy to implement, and the communicating parties can immediately interpret the content and meaning of the messages they exchange. However, this method of communication has a limited expressive capability.

The other approach is to create a general purpose formal language. These systems construct a computer language that is general in nature and able to express a wide range of concepts. The first advantage of this approach is flexibility, since these languages can express and collaborate on richer sets of problems. Also, being a formal language, it can be documented and used by different and disjoint teams building collaborative agents. Even though they share no code or common ancestry, they can collaborate using this common language. The DARPA Knowledge Sharing Effort (KSE) has developed a very capable formal language for intelligent systems to exchange information. In practice, both types of communication are used in collaborative agents.

5.3 Reasoning with Partial Information

Intelligent behavior can be loosely broken down into problem solving (planning) and reasoning on evidence. Several technologies are used to reason on partial information. This section will describe two common techniques: Fuzzy Logic and Bayesian Reasoning.

5.3.1 Fuzzy Logic

Fuzzy Logic was developed by Lotfi Zadeh for use in control systems that are not easily converted into mathematical models. He wrote a paper in the 60s called "Fuzzy Logic", which described this new technology. Acceptance in the United States was initially slow and may have been hindered by the word "fuzzy" in its name. Despite its slow start, it is now being used in a wide range of control systems.

Fuzzy Logic works well in domains that are complex and where knowledge is contained in engineering experience and rules of thumb rather than mathematical models. Fuzzy Logic allows the creation of set-membership functions that support reasoning about a particular value's degree of membership in that set. Figure 5.7 shows three fuzzy sets that define Cold, Warm and Hot. A specific temperature, say 100 degrees, has different membership in each of these sets. Combiners are used to connect fuzzy expressions together to generate compound expressions. Rules can be constructed that reason on these memberships and perform some action. For example, "if EngineTemp is Hot and the DeltaV is NecessaryDeltaV then Stop Burn" would reason on the current engine temperature in relationship to the necessary delta V and determine whether it should terminate burn. Other rules could deal with a hot engine and an insufficient delta V, or could stop the engine when the delta V is optimal.

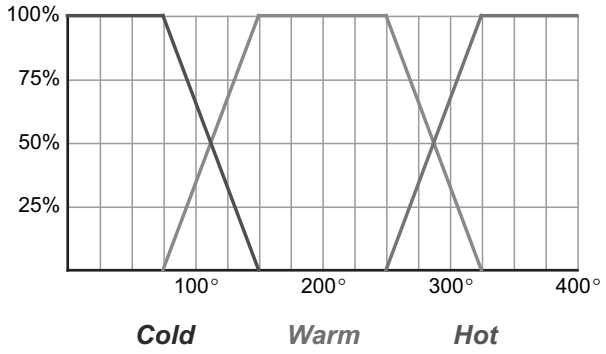

Fig. 5.7. Fuzzy Sets

Despite their name, Fuzzy Logic systems are time invariant, deterministic, and non-linear. They are computationally efficient. Once rules have been developed they can be "compiled" to run in limited computational environments. Unfortunately, how and when to use fuzzy combination rules is not well understood or systematic.

Engineers might be drawn to this technique because it is easy to get an initial prototype running and incrementally add features. If the domain is primarily engineering rules of thumb, the use of this technique is defensible. If, however, good mathematical or other models of the domain can be constructed, even at the cost of more up-front effort, the resulting system should be more robust. Serious consideration should be given to whether the initial ease of fuzzy construction outweighs the robustness of more formal modelling techniques.

5.3.2 Bayesian Reasoning

Bayesian reasoning is a family of techniques based on bayesian statistics. Its strength is the firm foundation of statistics on which it rests (hundreds of years). It has a well developed methodology for mapping real world problems into statistical formulations, and if the causal model is accurate and the evidence accurately represented, Bayesian systems give scientifically defensible results.

In simple terms, Bayes' rule (or Bayes' theorem) states that the belief one accords a hypothesis upon obtaining new evidence can be computed by multiplying one's prior belief in the hypothesis and the likelihood that the evidence would appear if the hypothesis were true. This rule can be used to construct very powerful inferential systems.

Bayesian systems require causal models of the world. These models can be developed by engineers using their knowledge of the system or in some cases by analyzing empirical data. These models must be at the appropriate level of detail for the system to make correct inferences. Multiple implementation strategies have been developed to reason with bayesian models once they have been created. Figure 5.8 (a) shows two methods for representing a bayesian model. Bayes nets encode the state information as nodes and causality as links between the nodes. Figure 5.8(a) shows a simple example. More recent work has developed systems that can reason on the probabilistic equations in a symbolical manner similar to Mathematica. An example data set can be seen in Figure 5.8(b).

Many implementations of bayesian systems make a simplifying assumption that the supplied evidence is completely independent from other evidence. This makes bayesian model generation and analysis much simpler. Unfortunately, the independence rule is often violated in real domains and distorts the final results. This effect has to be carefully controlled.

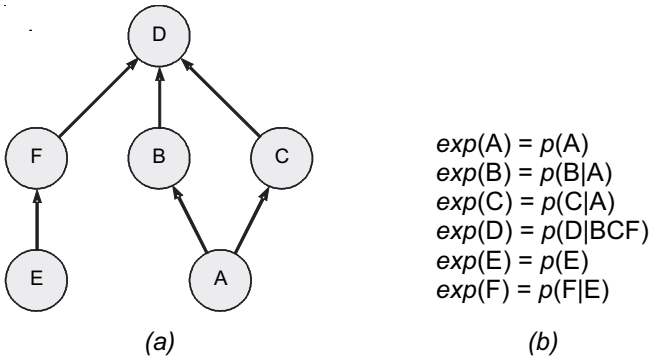

Fig. 5.8. Bayes network and symbolic representation.

5.4 Learning Technologies

Learning techniques allow systems to adapt to changing circumstances such as new environments or failures in hardware. They also allow systems to respond more rapidly to situations they have previously explored and to which they have found good solutions. While many techniques exist, this section will focus on Artificial Neural Networks and Genetic Algorithms.

5.4.1 Artificial Neural Networks

Artificial neural networks are a learning technique based loosely on biological neural networks. Figure 5.9 is a pictorial representation of a neural net. Each 'neuron' is a simple mathematical model that maps its inputs to its outputs. It is shown in Figure 5.9 as a circle. The input situation is represented as a series of numbers called the input vector that is connected to the first layer of neurons. This layer is usually connected to additional layers finally connecting to a layer that produces the output vector. The output vector represents the solution to the problem described by the input vector.

Neural networks are trained using a set of input vectors matched to output vectors. By iterating through this training set, the network is 'taught' how to map this and similar input vectors into appropriate output vectors. During this training, the network determines what features of the input set are important and which can be ignored. If the training set is complete and the network can be trained, the resulting system will be robust.

Neural networks have been used successfully in many domains. However, one difficulty with neural networks is that it is not always easy to understand what they have learned since the learned information is locked up in internal neuron states. A related difficulty is that there is no way to determine whether the training set was complete enough to have the network learn the correct

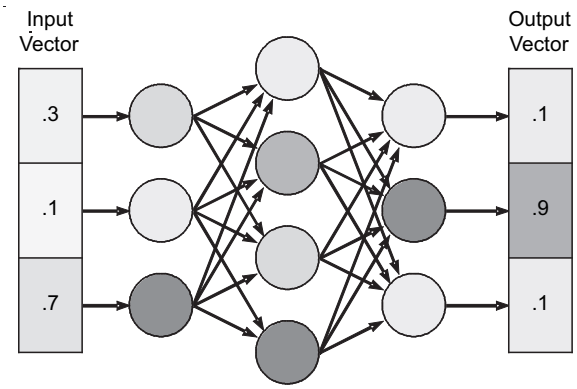

Fig. 5.9. Artificial neural network.

lessons. When faced with a novel situation, the network may choose the wrong answer. The final difficulty is that controlled adaptation can be good, but too much learning can cause a previously stable system to fail.

5.4.2 Genetic Algorithms and Programming

Genetic Algorithms are learning techniques based loosely on genetic reproduction and survival of the fittest. They use either numeric or symbolic representations for the definition of the problem.

Figure 5.10 is a pictorial representation of a genetic algorithm reproduction cycle. Processing begins by randomly creating the initial pool of genes. Each gene represents one potential solution and each is evaluated to determine how well it solves the problem. The best solutions from the current generation are paired and used to make new genes for the next generation. The bits from the two parents' genes are mixed to create two new genes. The new genes enter the gene pool to be evaluated on the next cycle. Since the best genes are continually being mixed together, the genes in later generations tend to be better at solving the problem. This cycle continues until a strong solution is found.

Genetic programs operate in a manner similar to genetic algorithms with the modification that each gene is actually a small program. Each reproduction cycle mixes parts of each parent program.

5.5 Act Technologies

The action portion of the autonomy cycle is responsible for implementing the choices made by the plan portion of the cycle. The actions interact rating environment and must be customized to the problem domain.

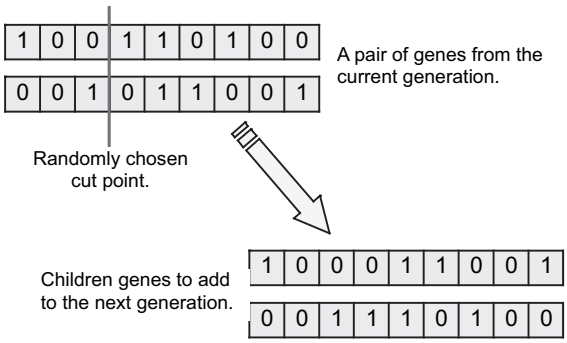

Fig. 5.10. Genetic reproduction cycle.

The most obvious action technologies are the actuators used by robots and immobot agents. Actuators are devices that are directly coupled in the operation environment and that are able to make some change when activated. Examples include opening a valve, moving a drive train, or starting a pump.

Being physically connected to the real world, actuators are subject to the wear and tear due to age and potential damage caused by the environment. Wear and tear can cause actuators to have complex failure pathologies, and robust systems are designed to detect these failures and recover. Since actuators modify the operating environment, designers must ensure that actuator actions do not have negative side effects for the agent or the environment.

Communication is a different form of action. Agents use it to share world views, negotiate options, and direct subordinates. Usually, the communication is transmitted over a network. The messages can be sentences in a formal language or the messages can be the exchange of computer data structures.

In software agents, all actions are represented as some form of communication to the subsystems being controlled by the agent. These communications cause the controlled system to begin some internal process or modify a process that is underway. Examples of these actions would include sending email to the owner, beginning a database search, cancelling a buy order, or sending a work request to another agent.

5.6 Perception Technologies

Perception is the activity of sensing and interpreting the operating environment. Like the action technologies, perception technologies are tightly coupled to the problem domain.

5.6.1 Sensing

Sensing is the process where some attribute or attributes of the environment are measured. Many types of sensors exist, from the simplest switch to complex multi-spectral image detectors, and they use a wide range of technologies to measure the environment.

In many cases, the sensor will take several steps to convert the sensed attribute into electronic signals that can be interpreted by computers. For example, a microphone converts the sound energy moving in the air into tiny vibrations in a mechanical system. These mechanical vibrations are then converted into electrical energy usually using piezo-electric or electromagnetic devices.

In some applications, the sensor is integrated directly into the actuator and helps the actuator achieve its desired effect. An example is the angle sensor in a servo system.

Sometimes the sensor and actuator are the same device. Electric motors, for example, convert electrical energy into mechanical rotation (actuator), but they can also be used to convert mechanical rotation into electrical energy for measurement (sensor). Another example is communication. The sending agent is performing an action and the receiving agent is sensing. When the receiving agent responds, the roles reverse.

5.6.2 Image and Signal Processing

To utilize the information supplied by the sensor, some sensor-specific computation is necessary. This computation is used to clean up the information supplied by the sensor and to apply processing algorithms that detect the attribute being measured.

Some sensors require little processing. A simple mechanical switch is either on or off and its interpretation should be easily understood by the agent. However, mechanical switches bounce when their position is changed, and this bounce causes a rapid series of on/off signals to be generated before the switch settles into its new position. The computer systems interpreting signals from switches are fast enough to detect these bounces and take them into consideration so as not to misinterpret the environment. Therefore, mechanical switches are de-bounced using a small electrical circuit or software. This is a simple example of signal processing.

Other sensors require massive amounts of signal processing. Radar and sonar systems work in environments with a large amount of background noise. If the signals are not carefully processed, the background noise will be incorrectly categorized as an item of interest to the system. Complex, sensor-specific algorithms are used to eliminate the background noise, interpret the results, and find target items in the signal.

In a similar manner, imaging detectors require large amounts of post-processing to clean up device specific noise and to interpret the data and

find target items of interest to the agent. In many situations, the algorithms are layered, where lower level algorithms clean up the individual pixels and higher level algorithms merge multiple pixels together to interpret what is in the image.

Image and signal processing can be computationally very expensive. Sometimes this computational expense will justify the use of special purpose processors whose only purpose is to perform the image or signal processing.

5.6.3 Data Fusion

The ultimate goal of perception is to supply the agent with an accurate representation of the working environment. In most real world systems, perception requires the agent to integrate multiple sensor data into a single consistent view of the world. This is often done by having each sensor perform its sensor-specific processing and then merge the sensor output [192, 92]. The techniques used vary between domains and the types of sensors being used.

One advantage of data fusion is that it can supply a more accurate understanding of the environment. This is because the data missing from one sensor can be filled in by another. Also, when multiple potential interpretations are possible in the data from a single sensor, information supplied from another sensor can help remove ambiguities.

Data fusion is a complex subject and many design decisions must be made when designing a system. Some of the important issues are:

- How is the sensed data represented? (pixels, numbers, vectors, symbols, etc.)
- How are the different sensor representations merged into the common view?
- How is conflicting information handled? Does one viewpoint win or are all views represented?
- Can uncertainty be represented with appropriate weightings?
- Can information from one sensor be used to fill in holes in the information from another?
- What is the level of granularity in the data (pixels, symbols, etc.) and how will the discrepancies between data granularity and model granularity be handled?

5.7 Testing Technologies

The development of robust agent systems requires testing. Testing supports the detection of errors in implementation where the system implementers overlooked or misunderstood a requirement, or just made a mistake. But another major purpose of testing is to uncover requirements that are missing in the original specifications when the system was designed.

A complete testing plan requires testing at each stage of the development effort and involves numerous strategies. Ultimately, the actual system (hardware and software) should be tested in a realistic environment that is capable of exercising both the nominal situations and many error cases. While this level of testing is important, it can be very expensive. As the cost of computation declined, it became possible to develop very realistic testing environments that exist solely in a computer without physical test hardware. This section will limit itself to software testing relative to cooperative autonomy.

5.7.1 Software Simulation Environments

Software simulation environments are based on the idea that it is possible to use computerized simulations to model accurately not only the system being tested but also its operating environment. While they cannot replace conventional testing of real systems, software simulations have many advantages. The two testing paradigms (i.e., real-world testing and testing in simulation) can be mutually complementary, and each can be used to cross-check the other.

Each of the testing paradigms has strengths and weaknesses that vary in different domains and problem situations. This set of relationships must be thoroughly considered and understood relative to the strengths and weaknesses of the development team. Only then can a cost-effective decision be reached as to the degree to which the development team utilizes each testing paradigm. NASA mission development teams have a long history of the use of both, and have a high level of corporate knowledge regarding the tradeoffs between the two paradigms for testing space mission systems. The expectation of the degree of cost-effectiveness of testing in a given mission development is strongly related to experience of this kind.

The first advantage, in certain situations, of software simulation environments is that the testing cycle time is shorter. Building real hardware and a physical simulation environment is, in some cases, very expensive. In contrast, again in some circumstances, once the software environment is set up, building models of the test system and its environment is relatively easy to do. This allows experiments in all levels of cooperative autonomy with only modest expense (depending on the problem domain).

As development proceeds, systems will need to be expanded, changed, or replaced. This, in many cases, may be very expensive to do when using physical test hardware. Software simulations, however, may allow these changes to be made quickly and cheaply. Indeed, in many situations, the costs are so low that several different solutions can be developed and tested without concern that the losing solutions will be thrown away.

Software testing environments also allow testing of system components in isolation. This allows components to be developed and tested without the need and cost of building a full model of the system. After testing, the promising components can be further developed and integrated into a complete system for further testing.

In a similar manner, software testing environments allow testing of different levels of fidelity. To perform a quick test of a new idea, it is not usually necessary to build a complex high fidelity model of the whole system. A simple, low fidelity model will usually tell the designer whether the idea merits further development and testing at a higher fidelity.

Software testing environments usually support superior debugging tools. Simulated systems can give the developer an ability to examine the state of the simulated hardware and software at any point in the execution. If an unexpected event occurs, the simulation can be stopped and examined in detail until the cause is determined. This reduces the testing time and allows the developer to have a better understanding of how well the system works.

Finally, software testing environments, in general, are repeatable and can be designed to systematically test for faults. Real systems must face many environmental challenges and they can fail in many different ways. Creating multiple environmental challenges with real hardware can become very time consuming and labor intensive. Software simulations can, in many circumstances, quickly and simply create challenges and test for faults. Also, depending on how the model is created, the software system may automatically create failure scenarios not imagined or tested by human engineers on real hardware. Often it is the unexpected faults that cause the most damage.

Crucial in the decision to adopt the approach of simulating the hardware and the environment is the question of whether the development team has the necessary understanding of the hardware and the environment. To design and implement simulation software that will have sufficient fidelity (i.e., that will be true to the real world) is, in many cases, fraught with difficulties. This issue must be considered with dispassionate realism and with the recognition of the failure of some past development efforts in both government and private-sector projects, due to an over-simplified view of the hardware, the environment, and the dynamics of the combination, and due to an overly optimistic view of the capabilities of the development team.

The most important design issues when creating software testing environments are:

- The goals of the simulation,
- The level of fidelity required,
- The types of debugging desired, and
- Required interactions with other simulations or software/hardware components

Though simulation software can exercise the flight software, the test software must always remain physically separate from the flight software so that it does not accidently get activated when deployed.

The rest of this section will examine the common techniques used to create software testing environments.

5.7.2 Simulation Libraries

Simulation libraries consist of a set of program library routines that the test component calls to emulate an appropriate effect in the test environment. The library routines emulate the effect of the calls, interact with the environment, and return appropriate information to the calling component. This type of environment works well for systems with well defined control interfaces.

Simulation libraries are the simplest testing environments to build because they only require the normal software tools and programmer knowledge to develop. Additionally, any software language and environment can be used for the simulation development. Simulation libraries tend to have low fidelity, particularly in the timing of hardware actions. Again, the simulation software would have to be kept separate from the flight software.

A good example of this technique would be testing a robot control system on a robot simulation. The control program would make the same calls it would on the actual hardware and the simulated calls move the robot in the environment and return the same sensor results the real system would return. If the simulation is good enough, the control system can then be placed in the real robot and run.

5.7.3 Simulation Servers

A more advanced technique is to use a separate simulation server for components being simulated and have the component being tested interact with the simulation server using a local area network. The primary advantage of this approach is that the simulation and test component run on different hardware and therefore each can have free access to needed resources. The simulation can be at an arbitrary and appropriate level of fidelity. If a high fidelity simulation is desired, very fast hardware can be used to simulate it. The software being tested can be run in an environment very close to the real system with the network interfaces being the only potential difference. If, for example, the planned hardware for the control software being tested is an unusual flight computer, then that exact computer can be used. It does not affect the simulation server's hardware.

A simulation server has an additional advantage: it can usually accommodate multiple simulated entities in the same simulated environment—a big advantage when testing cooperative autonomy systems. Though it is somewhat uncommon, software library techniques can be used in simulations accommodating multiple simulated entities.

5.7.4 Networked Simulation Environments

Another software simulation technique uses a network of computers to simulate multiple entities in a single shared simulation environment. The focus in

designing these systems is on efficient protocols that allow an accurate simulation in the shared environment. The primary advantage of this approach is that it allows very large and complex systems to be simulated.

The Distributed Interactive Simulation (DIS) system funded by DARPA and the military is an example of a networked simulation environment [6, 40, 48]. This very successful project built a training system that allowed hundreds of simulated entities (tanks, planes, helicopters, missiles, etc.) to interact in a realistic battlefield simulation. The thrust of this effort was originally to have human-controlled simulated hardware engage in interactions. The continuations of DIS have developed in many areas. Work relative to cooperative autonomy simulations is intended to bring an understanding of how real hardware and simulated entities can interact together in the same simulated/real world. Such an understanding would support the development and testing of real spacecraft or robots interacting with simulated ones in mutual cooperation.

6

Agent-based Spacecraft Autonomy Design Concepts

In this chapter, we examine how agent technology might be utilized in Flight Software (FSW) to enable increased levels of autonomy in spacecraft missions. Again, as stated in the Preface, our discussion relates exclusively to uncrewed assets (robotic spacecraft, instrument platforms on planetary bodies, robotic rovers, etc.) or assets that must be capable of untended operations (e.g., ground stations during "lights-out" operations). The basic operational functionality discussed in Chapter 2 is accounted for and allocated between flight and ground, and between agent and non-agent FSW. Those technologies required to enable the FSW design are touched upon briefly, and new autonomous capabilities supportable by these technologies are described. Next, the advantages of the design from a cost-reduction standpoint are examined, as are the mission types that might benefit from this design approach.

For each design concept, the appropriate distribution of functionality between agent and non-agent components, and between flight and ground systems, is discussed. For those functions assigned to remote agent processing, enabling technologies that are required to support the agent implementation are identified. Further, we determine for which mission types each design concept is particularly suitable, and describe detailed operational scenarios depicting the remote agent interactions within the context of that design concept.

6.1 High Level Design Features

The general philosophy of this FSW concept is that conventional, non-agent software will encompass all H&S onboard functionality. This FSW segment, referred to as the "backbone", will also contain functionality directly supporting H&S needs, such as slewing and thruster control, and will be directly accessible by realtime ground commands in the "normal" manner. The Remote Agent software will embody mission support functionality, such as planning and scheduling, as well as science data processing. To achieve its goals, a Remote Agent may access backbone functionality through a managed bus whose

data volume will be restricted to avoid potential interference with critical backbone processing. Should any or all Remote Agents "fall off" the bus or simply cease to operate, the backbone will not be impacted although science observations could well be temporarily degraded or terminated.

The FSW backbone processing is time-driven. Each function within the backbone receives a well-defined "slice" of processing time. All backbone functionality is scheduled to begin at a fixed time relative to the start of a processing cycle, and will complete within one or more time cycles. A list of backbone functions is provided below, with more detailed descriptions following:

1. Safemode
2. Inertial fixed pointing
3. Ground commanded thruster firing
4. Ground commanded attitude slewing
5. Electrical power management
6. Thermal management
7. H&S communications
8. Basic Fault Detection and Correction (FDC)
9. Diagnostic Science Instrument (SI) commanding
10. Engineering data storage

6.1.1 Safemode

Safemode is the key enabler of higher-level FSW functions. Safemode guarantees that no matter what may happen during the conduct of the mission, be it a hardware malfunction (for redundant H/W components) or a FSW abnormality, as a last resort (provided the problem is detected), the spacecraft can enter a state where no further permanent damage will be done and spacecraft H&S can be maintained. The allocation of this function to the FSW backbone is clearly essential.

Return from safemode is contingent on diagnosing the underlying cause of the problem and implementation of corrective action, either selection of a "canned" solution or creation of a new bespoke solution. Currently, return from safemode is within the purview of the Flight Operations Team (FOT), but, with a sufficiently advanced onboard FDC , the capability could be shared. Some spacecraft are provided with multiple levels of safemode, but nearly all GSFC spacecraft have a sun-pointing safemode to guarantee power and SI safemodes to protect delicate SIs. In the past, most GSFC spacecraft also had a hardware safemode in case the onboard computer (OBC) went down.

6.1.2 Inertial Fixed Pointing

Although the safemode function discussed above guarantees maintenance of spacecraft H&S, recovery from safemode and interception of the science schedule can be a lengthy process during which precious observing time will be lost,

potentially irretrievably depending on the mission lifetime and the nature of the science. So a lesser response to anomalies than safemode entry is highly desirable.

An inertial fixed pointing mode serves this purpose. For celestial pointing spacecraft, being inertially fixed is effectively being in observing mode without a science target and without making use of fine error sensor or SI data. The pointing accuracy and stability will therefore not meet mission requirements, but will be adequate to facilitate re-initiation of the science program. And virtually all spacecraft, independent of mission type, require an inertial fixed pointing mode to support sensor calibrations, such as gyro scale factor and alignment. For these reasons, and since ground controllers may need to command the spacecraft to an inertial mode either during the immediate post-launch checkout phase or in response to unusual spacecraft performance during mission mode, the FSW backbone will require the capability to transition to and maintain the spacecraft at an inertially fixed pointing. Note that typically the following associated functions are part of and subsumed under an inertial hold mode:

1. Gyro data processing and drift bias calibration
2. Attitude control law(s) for fixed pointing
3. Reaction wheel commanding and momentum distribution
4. Actuator commanding for momentum management

6.1.3 Ground Commanded Slewing

Just as ground controllers need access to an inertial fixed pointing mode in response to an anomalous condition on-orbit, the need may arise to slew the vehicle attitude to a different orientation. Control of nominal spacecraft slewing activities in support of science execution or sensor/instrument calibration could be allocated to an appropriate Remote Agent. But some basic slewing capability must also reside within the FSW backbone to ensure that in the event of the onset of an anomaly originating from within (or at least affecting) the agent itself, the backbone retains the capability for slewing the spacecraft to a needed orientation, such as sun pointing.

6.1.4 Ground Commanded Thruster Firing

Similarly to slewing, control of nominal thruster firing could be allocated to an appropriate Remote Agent. Nominal use of the propulsion subsystem might well be highly automated, coupled with autonomous onboard planning and execution of orbit stationkeeping maneuvers. However, the need will still exist to provide the ground controller a “back door” into the propulsion subsystem to enable ground commanding of emergency angular momentum dumps or orbit changes. For that matter, the FSW backbone itself may need to command the thrusters to damp out unusually high booster separation rates. So some access to and control of thruster firing by the backbone is critical.

6.1.5 Electrical Power Management

Even in conventional FSW designs, electrical power management is somewhat isolated from other FSW functionality to protect against "collateral" damage in the event of a processor problem or software bug. Some functionality within the electrical power subsystem may even be hard-wired. Although managing SI requests for use of electrical power might rightly reside inside a Remote Agent, management of the most critical spacecraft power resource must reside within the FSW backbone.

6.1.6 Thermal Management

As with electrical power management, thermal management in conventional FSW designs is already set apart from other FSW functionality, and for the same reasons. Also for the same reasons, it also must reside within the FSW backbone.

6.1.7 Health and Safety Communications

Although nominal communications should be highly autonomous to support efficient downlink of the massive quantities of data generated by modern SIs, in the event of a spacecraft emergency, ground controllers must be guaranteed direct access to any information stored onboard to help them understand and correct any problem. They must also be capable of sending commands to any hardware element involved in the resolution of the crisis. To support these fundamental mission requirements, especially during the early post-launch checkout phase, full ground command uplink capability (probably via a low volume omni antenna) must be provided both through the FSW backbone and through a command and data handling (C&DH) uplink card in the event the main processor(s) is (are) down. Low rate downlink through an omni must also be provided to guarantee the ground's reception of key status data, as well as (if necessary) more detailed engineering data stored just prior to the onset of a problem.

6.1.8 Basic Fault Detection and Correction

Just as all processing necessary for maintaining the spacecraft in safemode must be contained within the FSW backbone, the key functionality associated with transition into safemode must also reside within the backbone. In particular, basic FDC limit checking and safemode-transition logic should be part of the backbone. On the other hand, more elaborate FDC structures that adapt to new conditions, create new responses, or employ state-of-art methodologies such as state modeling, case-based reasoning, or neural nets should be hosted within an appropriate agent.

6.1.9 Diagnostic Science Instrument Commanding

If a SI experiences an anomaly within its hardware or embedded software, it will be necessary to downlink diagnostic data to the ground for analysis. In that event, as the SI or spacecraft platform may even be in safemode at the time, it is necessary to provide a "bullet-proof" capability for ground control to retrieve that diagnostic data, or even send troubleshooting commanding to the SI to generate additional information needed to solve the problem. To provide ground control a direct route to the SI for these purposes, the associated diagnostic SI commanding should also reside within the FSW backbone.

6.1.10 Engineering Data Storage

Just as communications with ground control could be fully autonomous and handled by Remote Agents (both on the spacecraft and in the ground's lights-out control center), so too could management of science data collected in the course of an observation. On the other hand, the main purpose of engineering housekeeping data is to support fault detection in realtime onboard, as well as anomaly investigations post-facto on the ground. To support these ground control efforts, which at times are critical to spacecraft H&S, the backbone should control storage and management of these data.

6.2 Remote Agent Functionality

The Remote Agents have the responsibility for achieving science mission objectives. They are event-driven, independent modules that operate as background tasks, which in some flight hardware architectures would be distributed among multiple processors. These mission-support agents are expected to negotiate among themselves and cooperate to accomplish higher level goals. For example, they might strive to achieve optimal science observing efficiency by coordinating the efforts of individual Remote Agents (potentially distributed between the flight and ground systems) responsible for data monitoring and trending, science and engineering calibration, target planning and scheduling, and science data processing.

To ensure the viability of the spacecraft, the FSW is designed such that individual Remote Agents can unexpectedly terminate functioning without impacting the FSW backbone. To isolate the agents from the backbone, an executive agent controls agent communication with the backbone. The bandwidth available for agent-to-backbone "conversations" is limited so as not to impact backbone processing. Bandwidth limitation is also used to control agent communications with spacecraft hardware through the backbone.

A list of potential Remote Agent functions is provided below, with more detailed descriptions in the following subsections:

1. Fine attitude determination

2. Orbit determination (and other reference data)
3. Attitude sensor/actuator and SI calibration
4. Attitude control
5. Orbit maneuvering
6. Data monitoring and trending
7. "Smart" fault detection, diagnosis, isolation, and correction
8. Look-ahead modeling
9. Target planning and scheduling
10. SI commanding and configuration
11. SI data storage and communications
12. SI data processing

6.2.1 Fine Attitude Determination

For safemode or inertial-hold purposes, gyros and sun sensors supply adequate information to guarantee acquisition and maintenance of a power-positive spacecraft orientation, or to ensure that the spacecraft will not drift far from its current attitude. So an autonomous onboard capability to determine a fine-pointing attitude (i.e., pointing knowledge good to a few arcseconds) is not critical to H&S. It is however essential for mission performance for any precision pointer, be it earth- or sun-pointer.

Given that "Lost-in-Space" startrackers are now available that directly output attitude quaternions, this function, for some missions, has already been realized in an independent hardware unit. For other missions with higher accuracy pointing requirements, FSW (that could be developed with an agent structure) would still be required that interpreted and combined data from the startracker with that from a fine error sensor or SI. The simplest implementation of this capability would involve creating a fine-attitude-determination agent that continuously generated attitude solutions, independent of the need of any other agent for their use. These solutions could then be stored in a data "archive" pending a request by other agents. Calibration data and sensor measurements needed by the attitude-determination agent could be stored in similar archives until requested by the agent. Old data (either input to or output from the agent) could be maintained onboard until downlinked, or if not needed on the ground simply overwritten periodically.

Orbit Determination (and other reference data)

Typically, accurate spacecraft position information is not required to support safemode processing or emergency communications, so orbit determination is not a function that must be resident in the FSW backbone. For spacecraft in near-earth orbits, GPS can be used for orbit determination. GPS also provides a time fix, simplifying onboard clock correlation. The GPS solution is purely a hardware one, constituting a fully independent, modular agent.

On the other hand, if future orbit information must be predicted, or if GPS cannot be used (as, for example with a mission at the L2 Lagrange Point), a software solution, often requiring input from the ground, must be used instead. The simplest implementation of this capability would involve creating an orbit determination agent that continuously generated orbit solutions, independent of the need of any other agent. These solutions could then be stored in a data "archive" pending a request by other agents. Similar to attitude data, old data could be maintained onboard until downlinked, or if not needed on the ground simply overwritten periodically.

In addition to the spacecraft ephemeris already discussed, there often is an onboard need for Solar, Lunar, ground station, and/or TDRS position information as well. These data are usually computed onboard via analytical models and would supplement the spacecraft orbit data already supplied by the orbit agent. In a similar fashion, other reference information such as geomagnetic field strength and South Atlantic Anomaly (SAA) entrance/exit times (for a set of SAA contours) could be supplied by the agent, as required by the mission.

6.2.2 Attitude Sensor/Actuator and Science Instrument Calibration

Currently, very few calibrations are carried out autonomously onboard. For nearly all current GSFC missions, gyro drift biases are calibrated at high cycling rates via a Kalman filter, and, for Small Explorer (SMEX) missions, onboard magnetometer calibration is standard. Neither of these is required to be performed (at least immediately) when in safemode or inertial hold, so they need not be part of the FSW backbone. In the future, however, the dynamic quality of other spacecraft hardware may require more elaborate onboard calibrations, both engineering-related and SI-related. In such circumstances, consolidating all calibration functionality within a single Remote Agent would facilitate interaction with (for example) a data monitoring-and-trending agent or a planning-and-scheduling agent, whether those agents are located onboard or on the ground.

6.2.3 Attitude Control

Other than for responding to ground realtime commands, spacecraft slews are performed in support of scheduled science observations or calibration activities. So, the attitude slew function can safely be assigned to a Remote Agent as long as a basic slewing capability is included in the FSW backbone as well. The agent version could be much more sophisticated than the version in the FSW backbone. For example, it could automatically adjust slew trajectory to avoid known constraint regions, while the backbone version followed a fixed eigenvector path. This same agent would have responsibility for high-precision fixed-pointing in science observation mode, a function also not required by the backbone.

6.2.4 Orbit Maneuvering

As long as direct thruster control is available to the ground via the FSW backbone, it is acceptable from a risk management standpoint to assign a higher level orbit maneuvering capability to a Remote Agent. This application would be responsible for planning routine stationkeeping maneuvers based on ground-developed algorithms.

To illustrate how the capability might be utilized inflight in a Remote Agent context, imagine that an autonomous onboard monitoring-and-trending agent has determined that a geosynchronous spacecraft will leave its orbital box in the next 24 hours. That agent would notify the planning and scheduling agent of the need for a stationkeeping maneuver in that timeframe. The planning and scheduling agent would request that the orbit maneuvering agent supply it with a package of thruster commands that will restore the orbit to lie within accepted limits. The orbit maneuvering agent, in part using, as input, data supplied by the trending agent, then constructs two sets of thruster commands, one to initiate a drift back to the center of the box and the second to stop the drift when the spacecraft reaches this objective. The orbit maneuver agent also specifies time windows within which both command packages must be executed, and submits its products to planning and scheduling, which schedules their execution.

6.2.5 Data Monitoring and Trending

Although rule-based limit checks for safemode entry need to be contained within the FSW backbone, more elaborate data monitoring and trending functionality could safely reside within a separate Remote Agent. This agent could utilize a statistical package to perform standard data analysis procedures such as standard deviation, sigma-editing, chi-squared, etc. to evaluate the believability of new measurements and/or calculated parameters and to extrapolate predicted values from past data. This agent would also be responsible for providing data products required by FDC to detect the presence of operational anomalies that do not threaten spacecraft H&S but could impact science data-collection efficiency or success. The output from this agent would also be of interest to application agents such as the orbit-maneuvering agent, as previously discussed. Note that the data monitoring-and-trending agent could also utilize a wide variety of AI products to generate and interpret its results, including state modeling, case-based reasoning, and neural nets, as well as a simulation capability.

6.2.6 "Smart" Fault Detection, Diagnosis, Isolation, and Correction

Some "primitive" fault detection driven by simple rule-based limit checking must be present in the FSW backbone to control entry into safemode and

other critical mode transitions and/or hardware reconfigurations. However that portion of fault detection solely concerned with threats to performance of science observations (as opposed to the hardware itself) would be assigned to a Remote Agent. That same agent could also perform fault diagnosis and isolation, and could determine the appropriate course of corrective action, which could be either a pre-determined "canned" solution or an original solution independently created by the agent. To the extent that fault diagnosis, isolation, and correction exist within the backbone, they should be viewed as canned correlations between simple limit checks and canned corrective actions developed and specified by ground personnel, primarily before launch. This agent may also utilize a wide variety of AI products to generate and interpret its results, including state modeling, case-based reasoning, and neural nets.

6.2.7 Look-ahead Modeling

The FSW backbone is a time driven software element that looks at data in realtime and responds in realtime or near-realtime: there is no need for any look-ahead modeling functions within the backbone. On the other hand, many Remote Agent applications (such as planning and scheduling, data trending and monitoring, and orbit determination and maintenance) may require the support of look-ahead models. Examples include ephemerides, solar intensity, wheel speed, and SAA entrance/exit. The model outputs could be generated continuously independent of the need of any other agent for their use. These solutions could then be stored in a data "archive" pending a request by other agents.

6.2.8 Target Planning and Scheduling

Traditionally, FSW has been time-driven, and full planning and scheduling responsibility rested with the ground system. The ground would generate an absolute-timed target list that the FSW would execute precisely as specified. If conditions were inappropriate for the science observation to execute (for example, if no guide stars had been found), the spacecraft would still remain uselessly at that attitude until the pointer in its C&DH FSW moved to the time of the slew to the next target. However, as planning and scheduling capabilities are migrated from the ground to the spacecraft, more flexible responses to anomalies are enabled leading to greater overall operational efficiency at reduced costs. It is these new capabilities, which are unnecessary to H&S maintenance within the FSW backbone, that are the purview of the target planning-and-scheduling Remote Agent. This agent may also utilize a wide variety of AI products, including state modeling, case-based reasoning, and neural nets.

6.2.9 Science Instrument Commanding and Configuration

Other than to support collection of SI diagnostic data and transition into SI safemode, no SI commanding and configuration functionality needs to reside in the FSW backbone. That functionality could be provided by a Remote Agent. The actions of the agent could be driven by receipt of template structures (generated by the planning and scheduling agent) defining what SI configuration is needed and/or what operational usage is desired. Separate agents could be assigned for each SI, or a single Remote Agent could handle the entire job. The agent(s) would also have responsibility for verifying the legality of any directives issued to an SI.

6.2.10 Science Instrument Data Storage and Communications

Although management of engineering data storage, as well as management of transmission of that data and SI diagnostic data, must be the responsibility of the FSW backbone, storage onboard and transmission to the ground of primary product SI data collected during a science observation is purely associated with satisfying science mission objectives and may therefore be entrusted to a Remote Agent. With a lights-out control center, one can easily conceive of all the duties associated with storing and transmitting SI data being handled autonomously by Remote Agents, where no general principle dictates whether these agents belong better in the ground system or the flight system. Instead, the decision on their location and relative distribution of responsibilities could, with a generalized flight/ground architecture, be made on the basis of simple convenience on a mission-by-mission basis.

6.2.11 Science Instrument Data Processing

Other than processing of SI data required to monitor the H&S of the individual SIs, no SI data processing need be contained within the FSW backbone. The advantage of assigning this functionality to a Remote Agent is that it enables cooperative behavior with other agents where the coupling of their functionality can yield a greater whole than the sum of their individual functions acting in isolation. For example, following collection of wide field data from a CCD detector in the course of a scan over a region of the celestial sphere, the data could be processed onboard by this agent, which (using a case-based pattern recognition algorithm) could identify point sources appropriate for more detailed study. The agent would report its results to the planning-and-scheduling agent, which would notify the slew agent to execute a return to the specified target coordinates. At the same time, the SI commanding-and-configuration agent, which prepares the SI, as necessary, for the upcoming observation utilizing plans generated by the planning-and-scheduling agent, would command execution of these plans at the proper time. The data generated would then

be processed by the SI data processing agent for storage and later transmission to the ground by that agent. By this means, immediate onboard response removes the need for scheduling revisits at a later date.

6.3 Spacecraft Enabling technologies

To support the Remote Agent functionality described in the previous subsections, a series of enabling technologies will be required. Many of these technology elements have already been flown on NASA missions, but have not as yet achieved mainstream status. Others are proposed for use on upcoming NASA missions, but are not as yet flight-proven. And others are still purely in the "talking" stage, but could reasonably be expected to be available in the next decade. The following is the list of technology items:

1. Modern "Lost-in-Space" star trackers
2. Onboard orbit determination
3. Advanced flight processors
4. Cheap onboard mass storage devices
5. Advanced operating system
6. Decoupling of scheduling from communications
7. Onboard data trending and analysis
8. Efficient algorithms for look-ahead modeling

The following subsection discuss these in more detail.

6.3.1 Modern CCD Star Trackers

The Wilkinson Microwave Anisotropy Probe (WMAP) mission (launched in 2000) utilized CCD star trackers containing both calibration and star catalog information, permitting the star tracker itself to output a direct measurement of its orientation relative to the celestial sphere. A simple multiplication (within the attitude control subsystem (ACS) FSW) by the device's alignment matrix relative to the spacecraft body yields the spacecraft's attitude quaternion. This capability enables fine attitude determination "on the fly" without *a priori* initialization, an important autonomy feature supporting calibrations, target acquisitions, communications, target of opportunity (TOO) response, and smart FDC.

6.3.2 Onboard Orbit Determination

Onboard orbit measurement via GPS is widely in use on commercial satellites and is planned for use on the Global Precipitation Measurement (GPM) mission (scheduled launch date of 2013). So, onboard orbit determination on the fly without *a priori* initialization is a capability readily available now for

spacecraft orbiting in the Earth's immediate vicinity. GPS timing also allows synchronizing the spacecraft clock with ground time.

The future challenge is to develop hardware and algorithms enabling fully independent onboard orbit determination far from the earth. Promising work along these lines is currently in progress, where star tracker measurements of celestial objects, such as the Moon, earth, and other planets could be used to derive spacecraft position information. Position information for the planets and the earth's moon are available now from self-contained analytical models requiring very infrequent (if any) input parameter updates. Independent spacecraft orbit determination is necessary to support other autonomous functionality, such as onboard calibrations, target acquisitions, communications, target of opportunity response, dynamic scheduling, and smart FDC.

6.3.3 Advanced Flight Processor

Nearly all the functionality projected in this design for implementation within a Remote Agent framework can be accommodated by flight processors that have already supported missions. However some capabilities require such abundant computing power (to support massive data processing and/or intricate logic analysis) that the development of a flight-capable, radiation hardened high performance computer may be required. These items may include areas such as onboard SI data processing, dynamic scheduling with look-ahead, and smart fault diagnosis and corrective plan creation.

6.3.4 Cheap Onboard Mass Storage Devices

Trends in this area are bringing costs (both dollar and weight) down so rapidly that in the future it is unlikely that onboard storage capacity will be a major limiting factor with regards to flight data processing capabilities. However, as SI data volume production is increasing rapidly as well, available storage will continue to play a dominant role in communications trade issues so long as all SI raw measurements must be downlinked, or lossless compression is utilized.

6.3.5 Advanced Operating System

Although no autonomy function discussed here absolutely requires an operating system with file manipulation capability comparable to a general purpose computer, such a capability would (for example) simplify ground handling of science observation downlink products, as well as the C&DH FSW's own manipulation of data onboard.

6.3.6 Decoupling of Scheduling from Communications

Decoupling of scheduling from communications has been beneficial in simplifying the Rossi X-ray Timing Explorer (RXTE) ground system's planning

and scheduling process and in reducing TOO response time. A similar decoupling could also be expected to make the job of onboard scheduling easier and enable a more dynamic and autonomous scheduling process onboard.

6.3.7 Onboard data Trending and Analysis

To support smart fault detection, diagnosis, and isolation, more elaborate capabilities for onboard data trending and analysis will be required. As discussed previously, a statistical package is needed to perform standard data analysis procedures (e.g., standard deviation, sigma-editing, chi-squared, etc.) in order to evaluate the believability of new measurements and/or calculated parameters and to extrapolate predicted values from past data. The statistical package will also be needed to support planning of new calibration activities.

6.3.8 Efficient Algorithms for Look-ahead Modeling

Conventional FSW typically is time-driven realtime software that tries to determine the best course of action at the moment as opposed to the optimal decision over a future time interval. This is an appropriate approach for standard FSW applications such as control law processing and fault-detection limit checking, but is not likely to yield acceptable results in areas such as planning and scheduling. For such applications, at least a limited degree of look-ahead modelling will be required to capture the complete information critical to efficient decision making.

6.4 AI Enabling Methodologies

To implement the Remote Agent functionality discussed in the previous section, more sophisticated modeling tools and logic disciplines will be required than the simple rule-based systems that have been employed in FSW in the past. In particular, a smart fault detection, diagnosis, isolation, and correction agent could also utilize a wide variety of AI products to generate and interpret its results, including state modeling, case-based reasoning, and neural nets. "Intelligent" constraint-evaluation algorithms used in planning and scheduling as well as the data monitoring-and-trending agent will also need these AI enabling methodologies. The following discusses onboard operations that would be enabled through the use of collaborative Remote Agents.

6.4.1 Operations Enabled by Remote Agent Design

Although the functionality assigned in the previous section to Remote Agents potentially could equally well have been implemented onboard in a more conventional fashion, a major asset of the Remote Agent formulation would have

been lost. If onboard implementations of these functions are evaluated as separate entities on an objective cost-benefit basis, for most GSFC missions one would probably find that more than half of those items could be developed more cost effectively in the ground system and will not significantly reduce operational costs if migrated onboard.

In particular, the following functions looked at in isolation should probably remain on the ground:

- Virtually all calibration, both science and engineering
- All data trending (although limit checking should remain onboard)
- Any smart fault detection, diagnosis, isolation, and correction
- Any look-ahead modeling
- All target planning; nearly all scheduling
- All specification of SI commanding and configuration (execution onboard)
- Nearly all communications decision making
- Nearly all SI data processing

The remainder already are largely onboard autonomous functions or in the near future likely will be.

However, an analysis of this kind misses a key aspect of the Remote Agent formulation, namely the ability of agents to engage in "conversations", negotiate, and collaborate. It is that distinctive nature of Remote Agents that makes their collective functionality more powerful than the simple sum of their component parts.

In the subsections that follow, examples are provided illustrating how the cooperative capacity of Remote Agents can yield more sophisticated (and more profitable) performance than they could achieve working alone. Operational capabilities enabled by the agents working together include:

1. Dynamic schedule adjustment driven by calibration status
2. TOO scheduling driven by realtime science observations
3. Goal-driven target scheduling
4. Opportunistic science and calibration scheduling
5. Scheduling goals adjustment driven by anomaly response
6. Adaptable scheduling goals
7. SI direction of spacecraft operation
8. Beacon mode communications
9. Resource management

6.4.2 Dynamic Schedule Adjustment Driven by Calibration Status

For spacecraft with high performance and accuracy requirements but very stable calibration longevity (both science and engineering), a ground scheduling system will be able to schedule science observations well in advance with a high degree of reliability. This confidence is due to the knowledge that should

calibration accuracy degrade prior to execution of a key scheduled observation, the degradation will occur gradually and gracefully, leaving the ground scheduling system ample time either to insert a re-calibration activity to restore nominal spacecraft function or to re-schedule any extremely performance sensitive observations for a later time.

However, if calibration stability is extremely dynamic, the "half-life" of the ground scheduling system's spacecraft state knowledge may be significantly less than the lead time on execution of many of the scheduled targets, in which case a pre-scheduled canned observing sequence will experience many observation failures and poor overall efficiency. An alternative to this is to couple realtime calibration status monitoring directly into planning and scheduling of both science observations and re-calibration activities.

For example, the ground could uplink to the spacecraft a target list having a label attached to each science target defining the level of telescope calibration needed to make the science observation worth while. The planning and scheduling agent could then elect to schedule only those list targets (or goal generated targets, as discussed later) compatible with the spacecraft's current state of calibration (as determined by the monitor-and-trending agent). After all such targets are exhausted, the agent could schedule a re-calibration activity (as created by the calibration agent) to bring the spacecraft back up to specifications, following which the remaining list targets could be observed. Alternately, utilizing another label supplying priority information, the presence of any target with a high priority designation could be sufficient to cause planning and scheduling to order a re-calibration activity immediately.

Although this example primarily illustrates the cooperative behavior of planning-and-scheduling, data monitoring-and-trending, and calibration agents, a somewhat lower degree of participation by several other agents (including attitude/orbit determination, attitude/orbit maneuvering, SI commanding-and-configuration, and SI data processing) may arise as a requirement.

6.4.3 Target of Opportunity Scheduling Driven by Realtime Science Observations

For most TOOs, the ground system, due to its access to data from other space and ground observatories, will be best positioned to designate appropriate TOOs for its spacecraft and adjust the observing schedule accordingly. Also, for those TOOs identified by processing SI measurements from the spacecraft itself, as long as those TOOs do not have a short lifetime and as long as spacecraft revisits are not extremely costly or inefficient, maintaining both SI data processing and TOO scheduling responsibility within the ground system is probably a better trade than migrating the capabilities to the spacecraft. However, if the TOO has a very short time duration (relative to turnaround times for ground SI data processing and scheduling), if target revisits are costly, or if the spacecraft is not in regular contact with the ground, advantage

can be gained by installing onboard at least limited functionality in these areas.

For example, after execution of high level survey measurements of a general region of the celestial sphere, an SI Remote Agent could process the data collected and identify point targets appropriate for the follow-up work. That agent could then inform the planning-and-scheduling agent of the presence of interesting targets in the immediate neighborhood, which could then schedule and execute those targets immediately (via SI commanding), avoiding the operational overhead of a re-visit following processing of the survey data on the ground. So by coupling the efforts of planning-and-scheduling and SI data-processing Remote Agents onboard, overall system responsiveness can be greatly improved and measurements of some classes of science targets can be performed that would otherwise not be observed. This capability has already been utilized on the Swift mission.

Although this example primarily illustrates the cooperative behavior of the planning-and-scheduling agent, the SI commanding-and-configuration agent, and the SI data-processing agent, additional participation by other agents (including attitude/orbit determination, attitude/orbit maneuvering, and SI data storage) could also be required.

6.4.4 Goal-driven Target Scheduling

Traditionally, the ground system has uplinked to the spacecraft a fixed, time-sequenced target list that the FSW has executed in an absolute time-driven fashion precisely as specified by the ground. This is probably the optimal scheduling approach for missions in which the science targets are easily defined in advance and the spacecraft is in regular contact with the ground. However, for missions more isolated from ground control or where the science targets can only be specified by general characteristics and have limited contact duration opportunities (e.g., asteroid flybys), a goal-oriented target-specification technique may be essential.

For the case of an asteroid flyby, the SI data-processing Remote Agent could not only perform straightforward data reduction of the measurements, it could also employ sophisticated pattern-match techniques (possibly via case-based reasoning) to identify regions for closer investigation. In this respect, the previously discussed TOO scheduling is just a subset of goal-driven target scheduling. However, goal-driven scheduling might also include the onboard specification of targets without any *a priori* ground input whatsoever. For example, suppose a percentage of a spacecraft's mission consists of performing survey work in selected regions of the celestial sphere. Knowing those portions of the celestial sphere targeted for survey work, the total percentage of time allocated to surveying, and guidelines regarding how much time should be spent on surveys within, say, a week-long interval, the scheduling agent could identify opportunities to "piggy-back" survey observations following successful ground-specified targets at approximately the same attitude.

6.4.5 Opportunistic Science and Calibration Scheduling

The previous discussion of goal-driven scheduling cited an example that could also be considered opportunistic scheduling, i.e., piggybacking a goal-specified survey observation on top of a specific ground-specified point target. However, opportunistic scheduling need not be goal-driven.

For example, while the spacecraft is executing attitude slews commanded for the purpose of acquiring targets and collecting science as specified by the ground, a calibration Remote Agent could keep a running tally of "miss" distances derived by comparing star-tracker-computed attitudes with gyro-measured angular separations. After collecting an adequate amount of data, the agent could calculate new gyro scale factors and alignments to maintain slew accuracy within performance requirements. The calibration computations would be done in the background on a non-impact, computational time-as-available basis relative to ongoing science or higher priority engineering support activities. Should representative slew data of a specific geometric variety be lacking, the calibration agent could request that the scheduling agent reorder the target list or consult its targeting goals to see whether a target could be "created" that would enable the collection of necessary engineering data as well. Failing that, the scheduling agent could simply add a slew of the required type to complete the calibration activity, if the fault detection and correction agent (using data provided by the monitoring and trending agent) indicated that a re-calibration needed to be performed in the near future to maintain spacecraft slewing accuracy requirements.

6.4.6 Scheduling Goals Adjustment Driven by Anomaly Response

This operational capability highlights cooperative behavior between the Remote Agents performing scheduling and fault-correction. Suppose that the review, by the fault-detection agent, of the fault monitoring-and-trending agent's spacecraft-state analysis has generated an error flag, which the fault-isolation agent has associated with a solar array drive. Suppose further that the fault-diagnosis agent has determined that continued nominal use of that drive mechanism will lead eventually to failure of the mechanism, and in response the fault-correction agent rules that the solar array should be slewed to its safemode orientation and not moved from that position until instructed by ground command.

At this point, the fault-detection agent could rule that all targets that previously were validated as acceptable and that required a solar array orientation other than the safemode position should now be considered invalid, thereby requiring that the scheduling agent delete those targets and at least compress the schedule, or possibly even generate a new one reflecting the currently degraded hardware state.

6.4.7 Adaptable Scheduling Goals and Procedures

A previous subsection discussed the cooperation of onboard Remote Agents to satisfy a ground-specified goal. In this subsection, we examine how the goals themselves might be modified/specified by the flight system based on inflight experience. Also, ground-specified procedures used for achieving those goals (either science observation or calibration goals) could similarly be modified. The input data for the goal modification process might be obtained from either of two sources, described now in the following paragraphs.

First, one could envision an onboard validation process that applies ground-supplied metrics to the execution of science observations, or, for that matter, engineering support activities. Sticking to the science application for simplicity, a monitoring and trending agent could calculate an overall observing efficiency, as well as individual efficiencies associated with the various different types of science (a function of SI, mode, target type, etc.). The fault detection agent might then look for relatively poor efficiency outliers so the fault diagnosis, isolation, and correction agent could determine which onboard goals or canned scripts/procedures might require enhancement.

Second, to define what changes might be made to those scripts/goals deemed inefficient, an onboard simulation function (under the control of the monitoring and trending agent) could be dedicated to running simulations on other approaches, either canned options provided by the ground or new options independently derived by the spacecraft (via ground-supplied algorithms) by running "what if" simulations in the background on a non-impact basis.

Although this capability would be primarily useful for missions where the spacecraft can expect to be out of contact with the ground altogether for entire phases of the mission (or during non-science cruise phases where most of the onboard computing power is idle), it might also prove useful for spacecraft constellations where several spacecraft themselves are cooperating to achieve overall constellation goals (see Chapter 9).

6.4.8 Science Instrument Direction of Spacecraft Operation

This section examines how a smart SI can influence the behavior of spacecraft platform subsystems. It is not unusual to allow SIs to command the ACS to adjust the attitude of the spacecraft to facilitate a target acquisition, or to shift the target from one SI aperture to another. For example, HST used SI specified "peak-ups" to facilitate realtime target acquisitions inside SI's having narrow FOVs. However the formalism of Remote Agents enables more elaborate interactions.

For example, suppose a spacecraft's SI complement included an instrument with a movable component (say, for survey scanning). Depending on the mass/motion of the component and the mission jitter requirements for fine pointing, the attitude perturbations induced by the component's motions

could threaten satisfaction of the jitter requirement. To deal with this potential problem, the agent embodying the SI could inform the attitude control agent of its intended motions in advance, so that the pointing control law could compensate for the disturbance in advance so as to protect a precision-pointing fixed-boresight SI observation from blurring. If the attitude agent determined it could not compensate for the effect, the fault correction and scheduling agents could be brought into the "discussion" to resolve the conflict, perhaps by giving priority to the precision pointing SI.

A more interesting solution would be for the more demanding SI to "announce" to an SI executive agent when it needed a particularly steady platform, and then have the executive agent prohibit motions by the lower priority scanning instrument during such periods. Similar restrictions would also apply to other moving structures, such as gimbaled antennas.

Another application for SI/agent direction of platform behavior is SI calibration. At its least elaborate, it would be straightforward to automate the process of periodically placing an SI in self-test mode (using its internal test source), comparing the output relative to a nominal signal, and passing on any discrepancy information to an SI calibration function. This could be a purely internal SI implementation. A more sophisticated example would involve periodic re-observations of a baseline target for the purpose of checking whether SI performance has remained nominal (and this example would entail agent collaboration with the attitude-control and scheduling agents). This example would also include the option for recalibrating the SI or Optical Telescope Assembly (OTA) if a significant degradation has occurred (where the re-calibration would be the job of the calibration agent).

6.4.9 Beacon Mode Communication

For missions where contact with the ground is expected to be irregular and where down-link requirements are driven by the spacecraft's need to inform the ground only of the results of science observations processed onboard (triggered by event messages from an onboard processing agent), it would be appropriate for the spacecraft to control communications. Similarly, if contact is driven by the spacecraft's need to confer with the ground regarding some problem experienced onboard (also triggered by an agent message), it would be most efficient/least costly for an onboard communications agent to "dial up" the ground's agent. In its extreme form, contact responsibility totally migrated to the spacecraft is referred to popularly as beacon mode, which can include periodic status summaries as well as the downlink of science measurement results and requests for help. Because the nature of beacon mode is to inform the ground of all onboard "experience" the spacecraft deems to be interesting to the ground, beacon mode operation can involve the interaction of the communications agent with nearly all the other onboard agents, including SI data processing, SI data storage (potentially part of communications),

data monitoring and trending, smart fault detection, diagnosis, isolation, and correction, etc.

6.4.10 Resource Management

Spacecraft operations concepts usually are driven by constraints imposed by limitations in onboard resources, both of the expendable varieties (for example, fuel or cryogen) and renewable varieties (such as electrical power, computing power, data storage, and telemetry bandwidth). When these resources are in demand by more than one distinct spacecraft subsystem or component, a mechanism must be provided (either ground-based or flight-based) to deal with the inevitable conflicts that will result. Other potential sources of conflict (which may also be thought of somewhat abstractly as resources) are priorities on use of onboard functionality, such as attitude control or thruster utilization/orbit adjustment. A Remote Agent formalism provides a particularly convenient infrastructure for resolving issues arising from overlapping needs.

In this application, one can envision a resources management agent (possibly subsumed under the executive agent) responsible for evaluating all resource requests in excess of a nominal set of limits associated with the set of agent users of those resources. The management agent would evaluate those requests relative to overall resource envelopes to determine whether the request can be honored, whether it must be rejected, or whether the need is high enough in priority that other agents must surrender a portion or all of their individual assets. The management agent would have to perform a similar function if an onboard anomaly unexpectedly reduced the availability of a resource. Due to the broad characterization of what constitutes a resource, resource management potentially can involve the interaction of that agent with all the other onboard agents. The management agent would also ensure that no resource deadlock situations arise.

6.5 Advantages of Remote Agent Design

The Remote Agent design described in the previous sections presents the opportunity for significant overall mission cost savings arising from its enabling of higher operational efficiencies, reduced FSW development costs, and reduced FSW testing costs. In the following subsections, each of these contributing factors are discussed.

6.5.1 Efficiency Improvement

Traditionally, onboard schedule execution has been mostly absolute time-driven. This design choice, while enabling an extremely simple onboard element in the overall schedule execution process, has necessitated the development of a very expensive ground planning and scheduling element run by

a large, expensive operations staff. The resulting product, due to the high reliance on the ground system's approximate look-ahead modeling with its associated worst-case time pads, has also been somewhat inefficient and quite limited in its capacity to respond to off-nominal or unexpected events. The following discusses cost savings enabled by an onboard scheduling capability.

Event-driven Scheduling

Autonomous, event-driven management of onboard activity transitions by Remote Agents has the potential for considerable cost savings. There always has been considerable use of event-driven transition control of onboard processes. For example, onboard logic can control mode transitions such as slewing to inertial hold when body rates and pointing errors have dropped below threshold levels, or the transition from inertial hold to safemode when anomalies are detected requiring immediate, extreme responses. Onboard logic, either existing within FSW or hardwired into the flight hardware itself, has also controlled transitions in sensors/SIs from search activities to tracking activities when the target objects' signal profile warranted it. Still, the use of an event-driven mechanism for managing transitions between scheduled science observations has often been precluded by the complexity of the spacecraft's orbital environment, performance demands imposed by the mission, and the computational limitations of the flight data system.

An example of an event-driven short-term scheduling system is the Adaptive Scheduler originally proposed for use on the James Wells Space Telescope (JWST). In Adaptive Scheduling's simplest form (from an onboard perspective), the ground system would still be responsible for generating an ordered list of desired science targets, but no absolute-time tags would be attached. The FSW then would observe the targets in the specified order, but nominally would trigger the execution of the next observation on the list in response to a FSW event signaling the successful completion of the previous observation. This avoids the waste of potential observing time engendered by the old paradigm's use of worst-case time pads to space out absolute-timed commands that might otherwise "collide" with each other.

The Adaptive Scheduler would also have the capability to insert into the timeline engineering events, as required, when informed by other FSW subsystems that an action needs to be taken. For example, when angular momentum has built up to the point that a momentum "dump" must be performed, the ACS would accordingly inform the Adaptive Scheduler, which would then find a good point in the timeline to insert the dump, and would order execution of the dump by the ACS at the appropriate time.

Further, in the event of the detection of onboard anomalies, the Adaptive Scheduler could take corrective action, which might involve deleting a target from the list, temporarily skipping a target until it could be observed at a later date, or even more elaborate reordering of the ground-supplied target list. For example, if after the ACS had exhausted all its acquisition options the JWST's

fine guidance sensor had still failed to acquire a guide star, the Adaptive Scheduler could command a deletion of that target from the observation list without waste of additional observing time and could order that a slew to the next target on the list be initiated.

For JWST's L2 orbital geometry, an event-driven scheduling system would be simple, efficient, and easily responsive to at least a substantial list of possible anomalies. A LEO orbit would present much more of a challenge due to the complexity of the environment, and would probably require the support of an elaborate look-ahead capability.

In one respect, adaptive scheduling is more complex than the absolute-timed paradigm. Because under the old paradigm, the start and end of onboard activities were very well defined, the process for insertion of realtime commands into the timeline (provided ample uplink opportunities were available) was well-defined, if at times awkward. However, under the new approach, ground operations staff won't always be certain when "safe" opportunities for realtime uplink might present themselves. Therefore, additional "intelligence" would need to be present onboard to enable the FSW to consolidate information and prioritize commands from a wide variety of realtime and preplanned/predicted sources, such as uplinked realtime commands, realtime sensor output, realtime event messages from onboard performance monitoring, and the ground-supplied target list.

The FSW must still support ground uplinks of time-tagged commands along with its list of event-driven activities for those situations where an activity must be performed within a specific time window (for example, a ground-planned orbit stationkeeping maneuver). This need is addressable via a short-term look-ahead capability that would support delaying scheduling any timeline events that would "swamp" the absolute-time window until after the absolute-timed event has completed, and giving the absolute-timed event priority over any routine engineering event that needs to be inserted.

It is apparent just from the Adaptive Scheduler example that considerable gains in efficiency can be realized just by moving from a fully ground-programmed, absolute time-driven style of operation to a ground-ordered, onboard event-driven style. Exploiting the onboard system's knowledge of realtime information enables a flexible response to on-orbit events that could greatly reduce loss of valuable observation time arising from conservative time-padding in ground look-ahead models. Non-replaceable onboard resources could be utilized more economically by expending them in response to realtime measured needs, while renewable resources could be allocated more appropriately and with smaller contingency safeguards or margins. Also, a side benefit to this new paradigm is that wherever expensive ground-system look-ahead modeling can be replaced by simple onboard realtime response, cost savings are achieved within the ground system as well.

The example cited in this subsection, the originally proposed JWST Adaptive Scheduler, is a rather modest effort at exploiting the potential offered by an event-driven FSW operation. The Remote Agent FSW formulation de-

scribed in this section would have much more powerful capabilities. More than just being an event-driven executor of a ground-planned, ground-scheduled (both long-term and short-term) target list, a Remote Agent design provides a framework for migrating short-term scheduling responsibility to the onboard system, greatly increasing the spacecraft's capacity for managing onboard resources more efficiently, reacting to anomalies without loss of observing time, and responding to TOOs while the opportunity presents itself.

Short-term Scheduling

Also, as discussed for the Adaptive Scheduler's event-driven scheduling, short-term scheduling by the FSW must be able to accommodate realtime commanding originating from the ground system. For onboard short-term scheduling, the complications will be even greater as there is no reason to expect the ground to be cognizant of ongoing onboard activities when the ground's command is sent. To deal with such intermittent interrupts of varying levels of priority, one can envision an onboard "fuzzy logic" reasoning module that juggles an array of priority levels and time windows attached to ordered/unordered target lists, target clusters, ground-originated realtime commands, ground-originated absolute-timed commands with/without windowing, onboard-originated housekeeping/engineering activities, realtime onboard sensor measurements, and externally-originated realtime data, and then deduces which activity (or activities) should be executed next (or in parallel).

Further, one could envision an event-driven Remote Agent engaging in a dialogue with more powerful computer resources (and models) on the ground to acquire deeper look-ahead information prior to making its activity transition decisions. The agent could even establish dialogues with other spacecraft to benefit from their realtime measurements. For example, if two earth-pointing spacecraft were flying in formation, the one arriving later over the target could interrogate the earlier one regarding cloud cover and other conditions and make appropriate SI reconfigurations prior to arrival.

6.5.2 Reduced FSW Development Costs

Conventional FSW is highly integrated code, optimized for timing performance and computational efficiency. As a result, long-term FSW maintenance tends to be very expensive (relatively small code updates may require very wide-spread regression testing) and FSW reuse is rather limited. However, a Remote Agent implementation offers much promise for producing significant reductions in FSW development and testing costs.

Since each agent can be built as a standalone module (consistent with the objects associated with object-oriented design), the development schedule for an agent can be synchronized with the availability of the information needed to define its requirements. As long as their interfaces with the executive agent and other agents with which they need to interact with are well-defined, those

agents (for example) associated with hardware components that will only be procured towards the end of the lifecycle can be developed later than those agents whose functionality is well-understood from the start.

As agents are developed, they can easily be added to the system, and if a problem develops with an agent inflight, it can be dropped offline without H&S impact to platform or payload. As this approach to developing FSW matures, it will be possible to build up a library of agents from previous missions, which can be reused economically on future missions once protocols and standards for agent communication have been established, eventually stimulating the creation of generalized COTS products, which will greatly facilitate the reduction of FSW development costs.

Significant reductions in FSW testing costs can also be expected. Since each applications agent is decoupled from direct communication with the FSW backbone and since their communication with the backbone is bandwidth limited by the executive agent, a modification to an applications agent should not normally require full-scale system-level regression testing. The modified agent could instead just be tested at a module level and then added back into the flight system. As an agent can drop offline without impacting the FSW backbone (and its corresponding H&S functionality), less stringent (and less costly) testing standards may be applied to the applications agents than to the backbone, and than currently applied to all conventional FSW. Finally, the similarity of this software architecture to typical ground system architectures should enable (in some cases) initial agent software development in the ground system, with the associated cheaper software development and testing methodologies, with later migration to the flight system following operational checkout (see Chapter 9 for a detailed example of this, called progressive autonomy).

6.6 Mission Types for Remote Agents

In this section, potential advantages of Remote Agents (over a conventional FSW design) are evaluated at a high level relative to a set of characteristic mission types, specifically:

1. LEO celestial pointers
2. LEO earth pointers
3. GEO celestial pointers
4. GEO earth pointers
5. Survey missions
6. Lagrange point celestial pointers
7. Deep space missions
8. Spacecraft constellations

The following subsections discuss these in more detail.

6.6.1 LEO Celestial Pointers

For LEO celestial pointers, the intermediate- and short-term scheduling problem is quite complex due to the many time-dependent constraints characteristic of near-earth orbits. The ground system component responsible for that function must be large and expensive, embodying many complex spacecraft and environmental models. But, since the dominant input to optimizing scheduling is not realtime measurements, there is little advantage in migrating short-term planning to the spacecraft, given that communications with the ground can be expected to be regular, frequent, and of long duration. It is even unclear that event-based scheduling would win a cost-benefit trade with conventional ground pre-programmed absolute time-based scheduling, given the need for look-ahead to maintain high scheduling efficiency. An exception to this general statement would be realtime support for TOOs, where the special onboard processing would not schedule the TOO itself, but instead simply make platform and payload housekeeping adjustments, as necessary, to support the change in the plan. So the planning and scheduling area probably is not a productive application for agent formalism for LEO celestial pointers, and, similarly, SI commanding and configuration is likely to be limited to carrying out ground instructions, probably via templates stored onboard to reduce uplink data volume.

On the other hand, one can easily imagine additional calibration functions being migrated to the flight system for LEO celestial pointers. For example, as miss-distance data from slews is collected onboard, the current state of calibration could be monitored autonomously and a background task could re-compute the gyro scale factors and alignments (supported by background fine-attitude and orbit-determination background tasks). Whenever fault detection determined that the current calibrations no longer were acceptable, fault correction would direct use of the current "latest-and-greatest" computed values. Implementation of each of these functional areas in the Remote Agent formalism via the design structure described earlier in this section would greatly facilitate the cooperative behavior described above without adding risk to the maintenance of platform and payload H&S via the FSW backbone.

Summarizing, implementing the following additional onboard functions as Remote Agents would be consistent with the lights-out control center philosophy:

1. Fine attitude determination (as discussed)
2. Orbit determination (as discussed)
3. Attitude sensor/actuator and SI calibration (as discussed)
4. Attitude control (execute slew requests; fine pointing)
5. Orbit maneuvering (plan and execute orbit stationkeeping)
6. Data monitoring and trending (as discussed)
7. "Smart" fault detection, diagnosis, isolation, and correction (as discussed)

8. Look-ahead modeling (probably not required)
9. Target planning and scheduling (not required)
10. SI commanding and configuration (execute science calibration requests)
11. SI data storage and communications (in cooperation with ground agent)
12. SI data processing (just for target acquisition)

LEO Earth Pointers

The scheduling problem for LEO earth pointers is much simpler than for LEO celestial pointers. Long duration look-ahead is no longer an issue since the spacecraft's orbit cannot be predicted to a high level of accuracy very far in advance. The planning aspect of the problem, however, is very target dependent, and might vary with time depending on science prerogatives. One can therefore imagine a set of templates (with ground tunable parameters) associated with individual targets (or target types) that an onboard scheduling agent could invoke whenever the onboard orbit-determination agent (a GPS receiver coupled with a short-duration orbit propagator) in conjunction with a data monitoring-and-trending agent deemed the target was coming into view. Because fuel available for stationkeeping maneuvers is directly equivalent to mission lifetime, it is unlikely that most NASA LEO earth pointers would expend fuel for large orbit-change maneuvers for the purpose of observing TOOs (short-duration events like volcanoes, for example), although the requirements on TOO response for some non-NASA spacecraft (for example, military imaging spacecraft) may be more demanding. Given that GPS receivers now give the spacecraft itself more immediate access to accurate, current spacecraft orbit data than does the control center, migrating some portion of the short-term LEO earth-pointer scheduling responsibility to the spacecraft to improve observational efficiency could be justified on a cost-benefit basis for some missions of this type, and migrating routine orbit-stationkeeping maneuvers to the spacecraft could reduce operations costs as well.

As in the case of the LEO celestial pointer in the previous section, one can easily imagine additional calibration functions being migrated to the flight system for LEO earth pointers. There also may be significant advantages in providing an SI data processing capability/agent onboard for the purpose of distinguishing between useful data-taking opportunities (for example, for a Landsat spacecraft, in the no-cloud-cover situation) from unusable conditions (in this case, a full-cloud-cover situation).

Summarizing, implementing the following additional onboard functions as Remote Agents would be consistent with the lights-out control center philosophy for LEO earth pointers:

1. Fine attitude determination (needed for orienting an imaging subsystem)
2. Orbit determination (as discussed)
3. Attitude sensor/actuator and SI calibration (as discussed)

4. Attitude control (needed for orienting an imaging subsystem; fine pointing)
5. Orbit maneuvering (plan and execute orbit stationkeeping, as discussed)
6. Data monitoring and trending (as discussed)
7. "Smart" fault detection, diagnosis, isolation, and correction (as discussed)

8. Look-ahead modeling (needed in conjunction with orbit-trending)
9. Target planning and scheduling (scheduling as discussed)
10. SI commanding and configuration (templates associated with targets)
11. SI data storage and communications (in coperation with ground agent)
12. SI data processing (as discussed)

6.6.2 GEO Celestial Pointers

A geosynchronous-earth-orbit (GEO) celestial pointer's operational constraints are much more straightforward than those for a comparable LEO mission, so much so that it is possible to operate the spacecraft in a "joystick mode", as demonstrated by the International Ultraviolet Explorer (IUE), where blocks of time were assigned to astronomers who could command the spacecraft directly when making their observations. This is not to say that automation is not critically important for enabling full utilization of spacecraft capabilities and performance accuracy at optimal efficiency, but that automation can be implemented on the ground in support of a human operator rather than migrated onboard to provide autonomous function. Since full contact with the ground station is nominally possible at all times and time delays in transferring spacecraft supplied data to the ground station are small, and because FSW development and testing costs are likely to remain significantly higher than those of ground software of corresponding size and complexity, this mission type is probably not a good candidate for an onboard Remote Agent implementation on a cost-benefit basis. However, Remote Agents may find many useful applications within the ground system's lights-out control center.

6.6.3 GEO Earth Pointers

As with the GEO celestial pointer described in the previous section, operational constraints are much more straightforward than those for a comparable LEO mission. And as before, full contact with the ground station is nominally possible at all times and time delays in transferring spacecraft supplied data to the ground station are small. Since FSW development and testing costs are likely to remain significantly higher than those of ground software of corresponding size and complexity, this mission type is probably not a good candidate for an onboard Remote Agent implementation on a cost-benefit basis. Again, Remote Agents may find many useful applications within the ground system's lights-out control center.

6.6.4 Survey missions

The nominal operation of a survey mission is by its very nature very straightforward and predictable. Planning and scheduling is well defined for extremely long durations. For a properly designed spacecraft, calibrations should be stable and easily monitored/trended by the ground. SI data need only be collected and dumped to the ground for processing and archiving; no onboard processing would be required, unless contact time and downlink bandwidth/onboard data storage availability is an issue. Again, in a cost-benefit trade, the relatively higher cost of FSW software over equivalent ground software will always be an argument in favor of a ground implementation decision (at least initially, with migration as an option). The one exception might be in the area of fault detection, diagnosis, isolation, and correction (supported by data monitoring and trending), where given the likelihood of non-fulltime contact between flight and ground, an autonomous capability to deal with a wide range of anomalies and still keep the mission going could prove very useful. Otherwise, it would probably be more cost efficient to maintain most potential Remote Agent functionality in the ground system.

6.6.5 Lagrange Point Celestial Pointers

Lagrange points are points of stable or unstable equilibrium relative to the influence of the combined gravitational fields of two celestial objects acting on a third object in orbit about the two objects. For simplicity of presentation, we'll restrict our discussion to the sun-earth Lagrange points, although Lagrange points can be defined for any two celestial objects close enough in proximity for both of their gravitational fields to affect a third orbiting object (for example, the behavior of the Trojan asteroids relative to the sun-Jupiter system). For a given celestial system, there will be a total of five Lagrange points, three (L1, L2, and L3) in-line with the two objects and two (L4 and L5) off-axis, as illustrated in Figure 3.2. Objects at the on-axis points (L1, L2, and L3) are in unstable equilibrium (i.e., objects at these points will drift off in response to perturbations), while those at the two off-axis points are in stable equilibrium.

The simple orbital geometry and benign environmental conditions (relative to those of a near-Earth orbit) make Lagrange-point orbits good choices for spacecraft with sensitive thermal constraints. The absence of occultations facilitates long duration observations of dim objects. In addition, for spacecraft in halo orbits about L1 or L2, the distance to the Earth is as low as a million miles, so communications-bandwidth needs do not demand excessively large antennas or transmitter power.

From a planning and scheduling perspective, the space environment is simple enough (contrary to the conditions at LEO) to enable considerable onboard scheduling autonomy. The absence of complex time-dependent constraints makes event-driven processing a preferred choice (over absolute time-driven, totally pre-programmed processing) for higher operational efficiency,

and in fact has been considered for use by the JWST mission. However, the value of onboard short-term scheduling is highly mission dependent.

Although communications with the ground can be regular, frequent, and of long duration, there still can be almost equally long periods when the spacecraft is out of contact. At these times, a smart fault detection, diagnosis, isolation, and correction capability, in conjunction with onboard data monitoring and trending, would (depending on designed redundancy capacity) allow the spacecraft to "fly through" a failure and continue its mission while out of contact with the ground. However, an equally viable approach would be simply to rely on conventional fault detection with a transition to a robust safemode to guarantee platform and payload H&S until contact with the ground is regained. Currently, the answer to the question of which approach is best is highly mission dependent and can only be determined following a rigorous cost-benefit trade. In the future, the answer will depend on how much progress has been made in standardizing onboard smart fault processing and in reducing FSW development costs relative to comparable ground software costs.

Similar arguments can be made for many of the other functions that might potentially be assigned to Remote Agents. Onboard SI data processing could be very helpful to opportunistic identification, scheduling, and acquisition of science targets (as well as TOO response). It could also enable a reduction in downlink bandwidth by telemetering only processed science products, not raw data, to the ground. But for some missions, the science targets are quite predictable and greater overall scheduling efficiencies may be obtained via a conventional ground scheduling system. And for most missions, the astronomer customer will insist on receiving the raw data rather than the downstream processed product. So again, an evaluation of the desirability requires rigorous cost-benefit analysis.

Summarizing, implementing the following additional onboard functions as Remote Agents would be consistent with the lights-out control center philosophy:

1. Fine attitude determination (needed for target acquisition)
2. Orbit determination (not needed)
3. Attitude sensor/actuator and SI calibration (need is a function of H/W design)
4. Attitude control (execution of slew requests; fine pointing)
5. Orbit maneuvering (infrequent; planned on ground)
6. Data monitoring and trending (as discussed)
7. "Smart" fault detection, diagnosis, isolation, and correction (as discussed)
8. Look-ahead modeling (a function of planning and scheduling autonomy)
9. Target planning and scheduling (as discussed)
10. SI commanding and configuration (execution of science and calibration requests)

11. SI data storage and communications (in cooperation with ground agent)
12. SI data processing (as discussed)

6.6.6 Deep Space Missions

Deep space missions are tailor-made for the flexibility and responsiveness enabled by a Remote Agent implementation of an autonomous spacecraft. Being out of contact with the ground for very long periods, and with significant radio-signal propagation time for communications, the spacecraft needs to have a greater degree of "self-awareness" not only to maintain H&S, but even to perform its mission efficiently. This is particularly true of a mission like an asteroid flyby where mission-critical observing decisions must be made in realtime. Although very complex deep space missions have successfully been performed by JPL in the past, JPL has rather clearly determined that the key to maintaining their recent downward trend in mission cost is to promote steadily increasing onboard autonomy through the use of formalisms such as Remote Agents. In the DS1 mission, the responsibility of health monitoring was transferred from ground control to the spacecraft [100, 135, 195]. This marked a paradigm shift for NASA from its traditional routine telemetry downlink and ground analysis to onboard health determination.

6.6.7 Spacecraft Constellations

Being recent in development, where deployment for now is largely restricted to communications networks, spacecraft constellations represent a new opportunity for flight autonomy applications and are covered in Chapter 9. Whereas consideration of Remote Agent applications to the previous mission types concerned interactions of spacecraft subsystems with each other or with the ground, for constellations the scale of interaction expands to conversations potentially between all the various members of the constellation. In a complex "conversation" of this type, just the job of determining which members of the constellation should be included in the conversation, when they should enter the interaction, and when they should drop off can be a thorny problem. More discussion on this topic will be provided later, but to support constellation interactions, a hierarchy of subsystem subagents controlled by agent spacecraft will need to be introduced.

6.6.8 Spacecraft as Agents

While spacecraft share many of the properties of the agents described above, the unique environment in which they operate makes unusual demands on their design. Since spacecraft are mobile, self-contained, and externally focused, they are often viewed as space-based robots, but this is not the complete picture.

While mobile, a spacecraft consumes much less of its time and resources for navigation than does a comparable robot. Navigation usually happens at only a few fixed points in the mission and the spacecraft is focused on other issues the rest of the time. In addition, the external orientation of a spacecraft is primarily for the use of science sensors whose data is usually shipped to earth and not used directly by the spacecraft. Most of the other sensors can be viewed as internally focused, distributed throughout the vehicle, and their purpose is housekeeping or health management of the craft. They perform activities to manage power, manage angular momentum, and keep the craft correctly positioned. This internal focus has led some to argue that spacecraft should be viewed as immobots. Certainly, some immobot technologies should be included in future spacecraft designs.

As autonomous spacecraft become more common, they will find themselves in the role of determining which science goals to pursue based on the current situation. If, for example, a pursued science goal cannot be met because of external events or an internal failure, the spacecraft will choose between the other available goals to maximize the science returned. Analyzing and prioritizing the information returned to humans is a primary area of research in software agents. In addition, software agents are a principal focus in the effort to build cooperative capabilities. In the future, it is likely that groups of spacecraft will work together to achieve larger science goals. These technologies, first worked out in software agents, need to be included in spacecraft designs.

While all of these agent technologies represent elements of the whole picture, NASA has the burden to evaluate them and adapt the technologies for spacecraft use. There are many ways that space-mission agent technologies differ from non-space agent technologies. Most software agents are ephemeral; their only goal is to acquire, manipulate, and exchange information, and the only resource they consume is computation. Spacecraft are not ephemeral. They exist in the real world and their primary resources are sensors and actuators. Actions consume tangible resources, and many of these resources are irreplaceable.

Actions in a spacecraft are usually costly, and once an action is taken, it may not be reversible. This makes it necessary for planning to factor-in the resources used and future cost carefully if the action cannot be retracted. These issues are usually ignored in software agents. Even most robots and immobots are not as deeply concerned about these issues as are spacecraft. Therefore, the planning and control systems of other agent technologies must be carefully studied and possibly modified for insertion in spacecraft systems.

Software agents assume that communication costs are modest and that communication is rapid. These systems often have gregarious agents that would rather ask for help than work out a solution for themselves. In space applications, most communication is expensive and the timeliness of delivery depends on the distance between the vehicles or between a vehicle and a

ground station. While collaboration is desired and necessary to achieve objectives, approaches that are more introspective will be necessary to limit communications and handle speed-of-light delays.

Much of the work on software agents has involved the construction of informational agents. Their primary purpose has been the acquisition of knowledge in support of human goals. While spacecraft both collect information and support human goals, to achieve their objective they must make many real-world decisions that require types of autonomy not necessary in the construction of informational agents.

When a software agent has a programming bug and fails, it is modified and restarted with little fanfare. This is often true of ground-based robots and immobots. Certain software failures in spacecraft are dangerous and could cause the loss of the whole mission.

Most believe that agent cooperation means communicating with humans or other agents to work towards a common goal. This is not the only mode of cooperation. Two or more spacecraft flying in formation may not be in direct communication with one another, but they still cooperate as the data they collect is merged to form a more complete picture. These simple modes of cooperation can assist in achieving missions with lower costs and risks.

As will be discussed in the next chapter, the potential for cooperative autonomy technologies in spacecraft systems is large but, as this section has shown, the use of this technology will require considerate and careful design.

7

Cooperative Autonomy

The philosophy of "faster, better, cheaper" reflected NASA's desire to achieve its goals while realistically addressing the changing environment in which it operated. One outgrowth of this philosophy is the shift from performing science missions using a few complex spacecraft to one where many simple spacecraft are employed. While simple spacecraft are faster and cheaper to build and operate, they do not always deliver better science. There must be offsetting compensation for any loss of power to deliver science value, by employing innovative technologies and new methodologies that exploit the use of less complex spacecraft. One such technology is cooperative autonomy.

Cooperative autonomy flows from the study of groups of individuals in terms of how they are organized, how they communicate, and how they operate together to achieve their mission. The individuals, in the context of space missions, may be human beings, spacecraft, software agents, or a mission operations center. From modeling and studying the cooperative organization and the interactions between its members come insights into possible new efficiencies and new technologies for developing more powerful space missions by which to do science more cost effectively.

Cooperative autonomy also creates new opportunities. Its technologies support cross-platform collaboration that allows two or more spacecraft to act as a single virtual platform and thereby possess capabilities and characteristics that would not be feasible with a single real platform. For example, with small telescope apertures on multiple, coordinated small spacecraft, the resultant aperture of the virtual combined telescope can be much larger than the telescope aperture on any single spacecraft.

This chapter outlines a model for cooperative autonomy. NASA's current mission organization is described and discussed relative to this model. Virtual platforms are also modeled and these models are used to assess the impact virtual platforms may have on the current NASA environment. Optimizations are suggested that could lower overall operations costs while improving the range and/or the quality of the science product. The optimizations could be

performed in a controlled and staged manner to minimize undesirable impacts and to test each change before the next is implemented.

The technologies for constructing cooperative autonomy systems will be discussed. Some of the technologies enable cooperation between humans and non human entities, while other technologies enable fully autonomous spacecraft. Many computer technologies are necessary, and each is at a different level of readiness for NASA environments.

7.1 Need for Cooperative Autonomy in Space Missions

As indicated above, cooperative autonomy is the study of how humans and computer agents can cooperate in groups to achieve common goals. This section describes some of the challenges faced by NASA in future missions, and informally describes how cooperative autonomy technologies can address these challenges.

7.1.1 Quantities of Science Data

Over the last decade or so, the rate at which science data is collected and returned by spacecraft has risen by several orders of magnitude. This is due, in large part, to advances in sensor and computer technologies. Data handling facilities have been expanded to handle the enormous amounts of incoming data, but the mechanisms supporting science extraction from the data have changed very little. In the current milieu of space missions, cross-mission data analysis is largely *ad hoc*, with large variations in data formats and media. The cost-effective ability to maintain scientific yield is in doubt without new approaches to support scientists in data analysis, reduction, archiving, retrieval, management, and correlation.

7.1.2 Complexity of Scientific Instruments

The instrumentation available to the scientist has increased, and continues to increase, in complexity, making it difficult to utilize fully the resources available. The amount of required documentation increases geometrically with complexity, and so the burden on investigators to become and remain able to effectively use any given instrument could easily get out of hand and become untenable for many investigators. Through experience, it has become clear that tools to map scientist's goals onto the space resources available to achieve those goals are essential.

7.1.3 Increased Number of Spacecraft

The number of active spacecraft has multiplied within the last decade. With the recent focus on small spacecraft, the numbers will soar. To realize the

potential of the small spacecraft, NASA must utilize them in associated arrangements, i.e., clusters, formations, or constellations. But to prevent a large growth in ground systems to accommodate the large number, they must be organized into virtual platforms that appear as a single entity to the ground. Virtual platforms, especially those that can be dynamically constructed, will allow science, formerly requiring huge spacecraft, to be performed with flexibility and effectiveness.

7.2 General Model of Cooperative Autonomy

The field of cooperative autonomy studies how autonomous agents should work together to achieve common goals. Autonomous agents can be humans, robots, spacecraft, or even factory shop floors. Cooperation can occur between any members of a group of agents, and their reasons for cooperation vary with their domain and mission.

This section will focus on describing a formal model of cooperative autonomy. Since autonomous agents play a central role in cooperative autonomy, it will begin by describing autonomous agents and discussing their properties. Once this groundwork is laid, cooperation between autonomous agents will be defined and four separate patterns for cooperative autonomy will be outlined. These four patterns are used to clarify the different facets of cooperation and describe individual attributes of cooperating autonomous agents. Most cooperative agents will simultaneously use two or more of these patterns.

7.2.1 Autonomous Agents

All autonomous agents share a set of common features independent of their operating domain and individual skills. These common features are outlined in Figure 7.1 (based on Figure 5.1 in Chapter 5) and they can be briefly summarized as: the ability to make plans, to act upon these plans, and to perceive and internalize the external world.

The most important single feature of an autonomous agent is its ability to plan. Planning is the process of making independent choices based on internal goals and the agent's beliefs about the operating environment (see Chapter 5). The results of this process are plans of actions to perform. Plans are the primary mechanism used by an agent to pursue its goals. Without the ability to plan, an agent would hardly be considered autonomous. To be considered a rational and intelligent agent, it must also adapt its plans to the changing situations in the operating environment. The agent's set of beliefs about the operating environment is commonly called its world model. Some agents also model their own internal state (self models).

Agents must act on the plans they make and these actions change the operating environment. Actions vary greatly from domain to domain and are always customized to the problem being solved. They are in some sense the

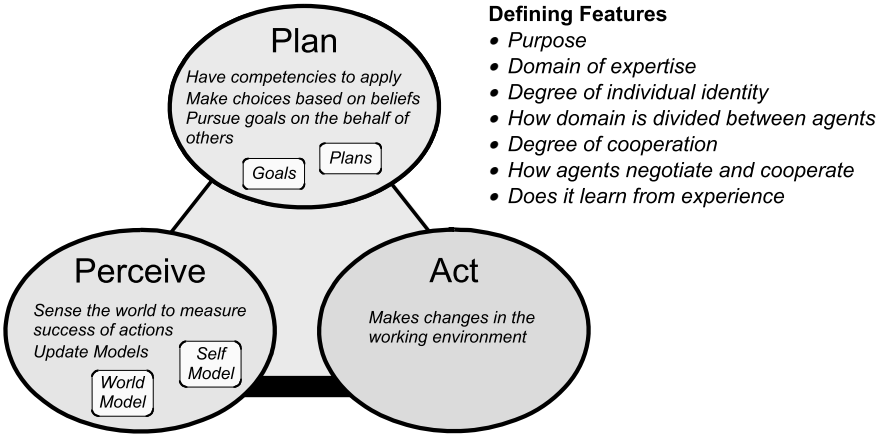

Fig. 7.1. The Features of Autonomous Agents. The common features can be seen in the diagram on the left and they are the ability to plan, to act, and to perceive the external world. The feature list on the right describes the unique characteristics of an autonomous agent that define its purpose and necessary skills.

final manifestation of the agent's plan and without them the agent's choices have no meaning.

The ability to perceive the operating environment is another feature common in all autonomous agents. Its primary purpose is to allow the agent to determine whether previous actions were successful and to detect changes in the operating environment. This information is used to update the agent's world model as well as its self model and ultimately to allow the agent to adapt its plans in the continuing pursuit of its goals. In some agents, the sensing of the operating environment is a goal in itself and this sensed information is delivered to the agent's superior without interpretation.

While the abilities to plan, act, and perceive are common to all agents, there are many features where they may differ. Some of these features are listed on the right of Figure 7.1. An agent's purpose and its domain of expertise are the defining aspects of an agent. Other key aspects include: what the agent must do, what knowledge and skills it must have, the actions it must be able to perform, the kinds of perception required, and how it plans.

The domain and purpose dictate the degree of individual identity the cooperating agents must have. In some domains, the cooperating agents have

common skills, share control of the resources, and have large overlaps in their world models. Often the world models are managed externally to the agents. Agents working in these domains tend to have a low degree of individual identity and are often interchangeable. They have little or no self model. An example would be a large public scheduling task where the agents are all working together to create a common schedule. The schedule, which forms a large part of the world model and represents the use and control of resources, is external to the agents performing the scheduling.

In other domains, the individual agents have direct control of resources and often they do not share much of their world model with other agents. Such agents tend to have a high degree of individual identity and when cooperating, the individual agents must negotiate as peers to achieve their common objectives. Their self model is expansive and it represents the status of all resources and systems under their control. Spacecraft are agents with a high degree of individual identity since they manage unique resources in unusual places. For example, multiple spacecraft attempting to perform joint science would need to coordinate their positions, their orientations, and the times when they need to perform actions such as data collection.

In some domains, some goals can be achieved with only sporadic cooperation, while others require continuous contact as the execution proceeds. The previous spacecraft scenario is a good example of sporadic cooperation to collect the necessary science. High precision formation flying is an example of continuous cooperation since each spacecraft must constantly sense and modify its position in relationship to its neighbors. Some domains have a specific hierarchy of responsibility, with the lower agents subservient to the upper. Human controllers of a spacecraft demonstrate this hierarchical cooperation. The controllers make the high level decisions, which are communicated to the spacecraft as the lower level agent for execution. In other domains, agents are direct peers who work together to come to a common agreement.

Finally, agents differ in how well they learn from experience. Some systems have fixed, prescribed rules to specify how the agent should operate under all known situations. Most current spacecraft fit in this category. Other systems learn as they operate. These systems can adapt to changing environments and over time can become more skilled. Human agents are a good model of learning agents. With the attributes of autonomous agents described, it is now possible to examine the behavior of groups of autonomous agents. This is the topic of the next section.

7.2.2 Agent Cooperation

There are a number of software-related aspects of cooperative autonomy that are embodied in agent technologies, which is a broad field. Figure 7.2 (see Chapter 5) shows an overview of agent technologies and the lower level technologies that are used to construct agents.

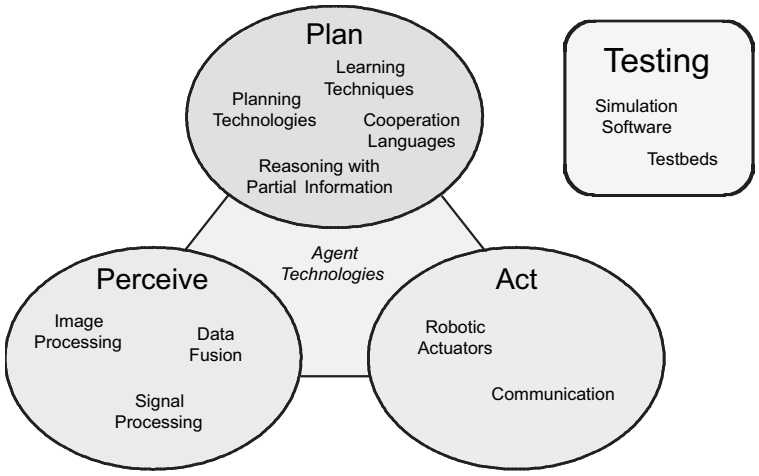

Fig. 7.2. Cooperative autonomy technologies.

Computer-based agent technology is an active area of research and its goal is to build computerized autonomous agents that fit within the models defined in Section 7.2.1. Agents cooperate with one another in different ways. In the simple case, agents work alone to achieve their goals. These types of agents usually assume that no other agent is in the environment, and their plans can be generated and pursued without concern of interference from others. Sometimes, multiple agents work cooperatively to achieve a common goal. When designing these agents, issues such as how the domain is divided between agents, how the agents negotiate among themselves, and the degree to which they cooperate must be resolved.

Cooperation occurs whenever autonomous agents work together to achieve common goals. Cooperation can occur in a variety of ways depending on the domain and goals being pursued. The following sections will describe several different patterns of cooperation and show them in relationship to the Plan, Act, and Perceive cycle described above. These patterns are not mutually exclusive, and often the agents must simultaneously engage in multiple cooperation patterns.

Cooperative Planning

Autonomous agents that work in groups as peers use cooperative planning. They cooperate with one another to reach agreement on the actions each member should perform to achieve their common goals. Figure 7.3 depicts a group of autonomous agents that are engaged in cooperative planning. Each layer of the Plan, Act, and Perceive cycles represents a separate autonomous

agent and the dots represent the points where these layers communicate during cooperation. In cooperative planning, the communications are focused on exchanging local views of the world, local and shared goals, actions that can be performed, and shared priorities. During cooperation, each member must reach an agreement on the set of actions that it will perform. In this pattern, the agents exchange information during the planning activity and before action is taken.

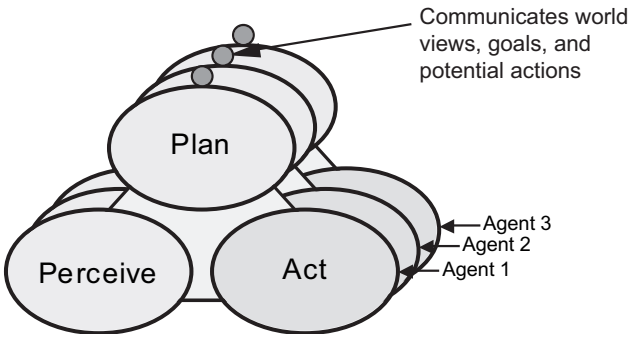

Fig. 7.3. Cooperative Planning.

This form of cooperation is peer to peer. No single agent needs to be in charge and yet each agent participates in a global negotiated solution. This pattern of cooperation occurs on a daily basis in human activities. The ubiquitous "weekly status meeting" is an opportunity for a group of people to share the results of the previous week's activities, discuss individual and group priorities, and to agree on the activities of each member for the coming week.

Hierarchical Cooperation

Hierarchical cooperation occurs when the agents have specific responsibilities and authority. In hierarchical cooperation, the superior agents decide the overall strategy and goals the subordinate agents are responsible for achieving. The subordinate agents will in turn plan and execute based on their local view of the domain and they will report their successes and failures to their superiors.

Figure 7.4 shows one example of hierarchical cooperation. In this example, the superior agent plans the activity of the group and its Act step is to communicate the plan to the subordinates. The subordinate agent receives the plan as a series of goals. These goals are interpreted from the agent's perspective and a plan is generated and executed. The subordinate agent then

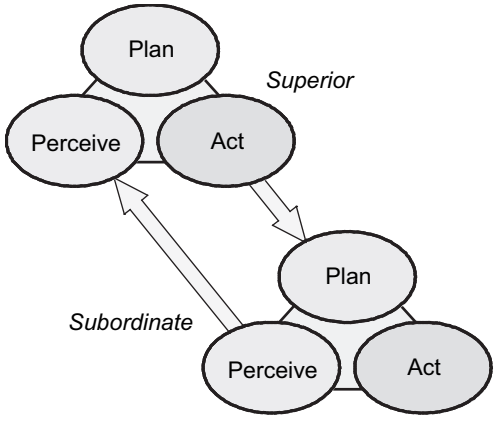

Fig. 7.4. Hierarchical Cooperation.

senses its local environment to determine how to continue the pursuit of its goals.

Hierarchical cooperation is not peer to peer. While a subordinate can negotiate with its superior by communicating its goals, resources, and constraints, it does not make the choice on what will be done. Once the superior has made these choices, the subordinate must attempt to achieve the goals to the best of its ability.

Hierarchical cooperation can be found in many business organizations. The superior agent in Figure 7.4 could represent a senior management group and the subordinate agent a department within the company. The senior management group sets goals and priorities for the department. The department manager, usually at department status meetings, distributes these goals in the department. The department manger also brings back the results of the department to the senior management group. In this way, the perceptions of the department become part of the perceptions of the senior management.

Computerized systems allow the connection between hierarchical layers to be much tighter than is possible in human cooperation. In these systems, the two levels are directly connected together with the action of the superior agent directly converted into plans for the subordinate agent. Figure 7.5 depicts this scenario. An excellent example of this type of hierarchical cooperation can be found in many robotic control systems. An upper level agent is a slow and deliberative planner that determines the overall strategic goals. The subordinate agent is a high-speed reactive planner. This planner converts the high level goals into direct robotic actions and responds rapidly to changes in the environment.

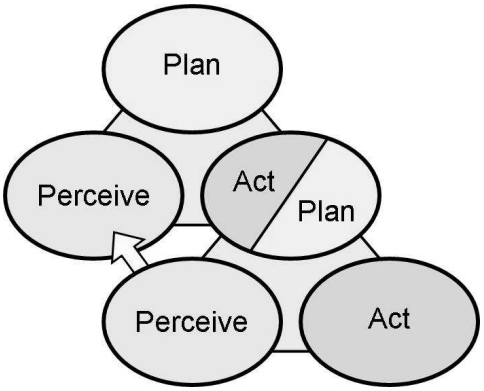

Fig. 7.5. Tightly Coupled Hierarchical Cooperation.

7.2.3 Cooperative Actions

Cooperative actions result when autonomous agents work together to achieve the desired objective. Tightly coupled actions result in a form of cooperation depicted in Figure 7.6.

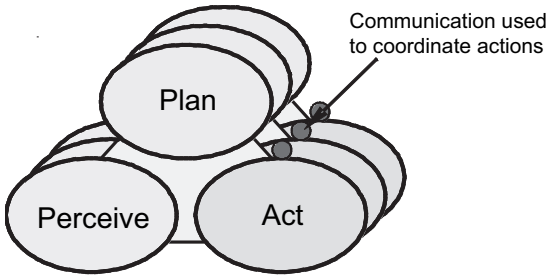

Fig. 7.6. Cooperative Actions.

Cooperative actions are usually coupled with other forms of cooperation. Satellites flying in formation to collect coordinated imagery are an example of cooperative actions. The individual vehicles must cooperate to keep themselves in the proper position and orientation with respect to each other. However, to achieve this high level cooperation, there must be another lower level cooperation that actually keeps the vehicles in formation (possibly some form of closed loop control).

Cooperative Perception

Autonomous agents use cooperative perception when they need to fuse their individual perception information into a single common perception. This process is commonly called data fusion or multi-sensor fusion [92, 192]. Many different techniques have been developed that fuse either similar or dissimilar kinds of information into a common model of the perceived world. This form of cooperation is depicted Figure 7.7.

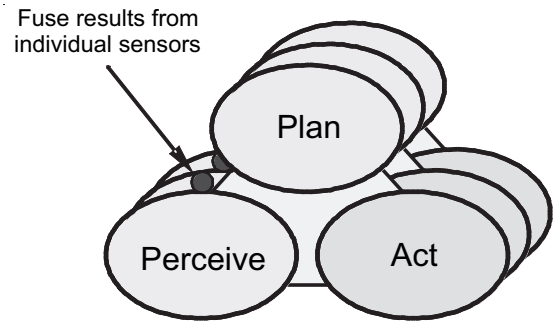

Fig. 7.7. Cooperative Perception.

A good example of cooperative perception is when imagery is collected on the same region of the earth in different spectral bands. When these different views of the region are merged into a single common view, the resulting data is more than the sum of the parts. Another example is the advisors an executive may talk to before a difficult decision. Each advisor has different perceptions of the situation at hand and the consequences of any particular action. The executive weighs the individual contributions and makes a decision.

This completes the discussion of a formal model for cooperative autonomy. An overview of current spacecraft mission management will now be offered to provide a foundation for discussing the application of cooperative autonomy technologies.

7.3 Spacecraft Mission Management

Spacecraft mission management is a complex process involving the coordination of experts from diverse disciplines. This section will outline a spacecraft mission model and describe the attributes of each group in the process. While it mixes some of the nomenclature from the deep space and earth science domains, the model provides a generic view of mission management organizations.

Figure 7.8 is a graphical representation of the management of a typical spacecraft mission. While the organization for a specific mission may vary, generally five different activities make up mission management. The process begins with the Science Planning group, which is responsible for creating the science plan representing the science goals. Using the science goals and spacecraft-housekeeping constraints, the Mission Planning group creates the overall mission plan for the spacecraft. This plan is passed to the Sequence Planning group, which converts the high level plan into a series of individual commands that the spacecraft will perform. This command sequence is passed to the Command Sequencer, which uplinks the commands to the spacecraft and monitors the results. Finally, the spacecraft delivers science data, which goes to Science Data Processing to convert it into a form useful to scientists. At various points in the process, telemetry or science data provide feedback to allow one or more of the planning groups to change direction.

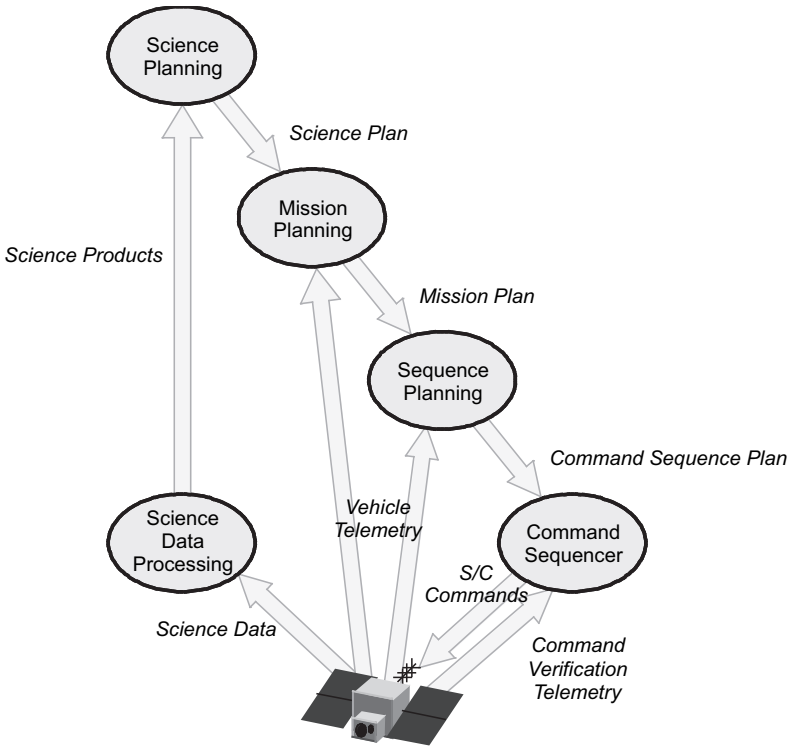

Fig. 7.8. Spacecraft Mission Management.

This section now describes each group, including its responsibilities and products.

7.3.1 Science Planning

Science planning for typical missions involves a large group of scientists, mission engineers, and instrument engineers with different backgrounds, domains of expertise, and modes of operation. This makes the science planning activity an exercise in collaboration and negotiation. A milestone in this process is the production of the science plan, which details the science goals to be achieved in the order of their importance or priority. There is little automation assisting the generation of the science plan. The product of this group is typically a text document that is provided to the science team.

7.3.2 Mission Planning

The Mission Planning group is a team of mission and instrument engineers that converts the science plan into the mission plan. The mission plan is designed to achieve the science goals while ensuring the health and safety of the spacecraft. The non-science activities include maneuvering and housekeeping, as well as scheduling time on a communications link. Mission planning usually requires one or more domain experts for each spacecraft activity. After planning the mission, these experts monitor the spacecraft for safety and health violations and adapt the mission plan as necessary. This group meets more frequently than the science planning group, because their collaborations are focused on producing a very detailed, usually short-term, mission timeline. The product of this group is either a text document or an electronic schedule of mission activities.

7.3.3 Sequence Planning

An individual or small group of mission engineers produces a very detailed command sequence plan using the mission plan as a guide. This plan specifies all of the commands and communications that take place between the ground and the spacecraft. For a typical mission today, the sequence plan is a detailed timeline for every low-level command to be uplinked to the spacecraft.

7.3.4 Command Sequencer

The Command Sequence software uplinks files of commands to the spacecraft, receives down-linked telemetry and information on spacecraft anomalies and verifies that the file of commands was uplinked with no errors. In case of an uplink or command failure, commands may just be skipped, or sequence planning may be repeated, or the spacecraft may be put in safemode. If an anomaly occurs, depending on the severity, replanning may have to be done after the anomaly is resolved.

7.3.5 Science Data Processing

Science Data Processing converts the raw data down-linked from the platform into useful science data for dissemination to a wide audience. This processing is conducted on the ground and often involves massive amounts of processing and data storage. The Science Planning group, especially the scientists, constantly monitor the science data produced by the mission. Based on the science data produced, the Science Planning group will sometimes modify the activities and priorities for the mission to produce an updated science plan.

7.4 Spacecraft Mission Viewed as Cooperative Autonomy

In this section, we combine the cooperative autonomy model in Section 7.3 with the spacecraft mission model of Section 7.3. Figure 7.9 shows the current mission organization, now redrawn from Figure 7.8 in terms of what it might look like with hierarchical cooperating groups. The hierarchical cooperation pattern is well suited to describing the spacecraft mission organization. The results of this combined model are discussed below.

7.4.1 Expanded Spacecraft Mission Model

At each level of the hierarchy in Figure 7.9, a group is focused on a specific domain and its purpose is defined by its position in the hierarchy. For example, the domain of the science planning group is concerned with the science aspects of the mission. Their purpose is to produce a science plan that maximizes the science returned by the mission. Each layered autonomous cycle in the science planning section represents one member of the planning team (there are four in the figure).

The diagram shows that these members are simultaneously engaging in each of the possible cooperation patterns. The most important of these is the planning collaboration between members, which sets the science goals for the mission. It is also used to evaluate the science returned and to adapt the science plan to maximize results. The science planning group must also cooperate during the action phase when the members work closely together to generate a single science plan that represents their views. This plan is then communicated to the mission planning group. The relationship between the science and mission planning groups is hierarchical in nature and represents the third cooperation pattern shown. The final cooperation pattern is the data analysis, in which the group must fuse all the data coming from the spacecraft to generate a single coherent view of the science being returned. Sometimes the science being returned will cause the teams to reevaluate their plan and adapt it to collect additional information.

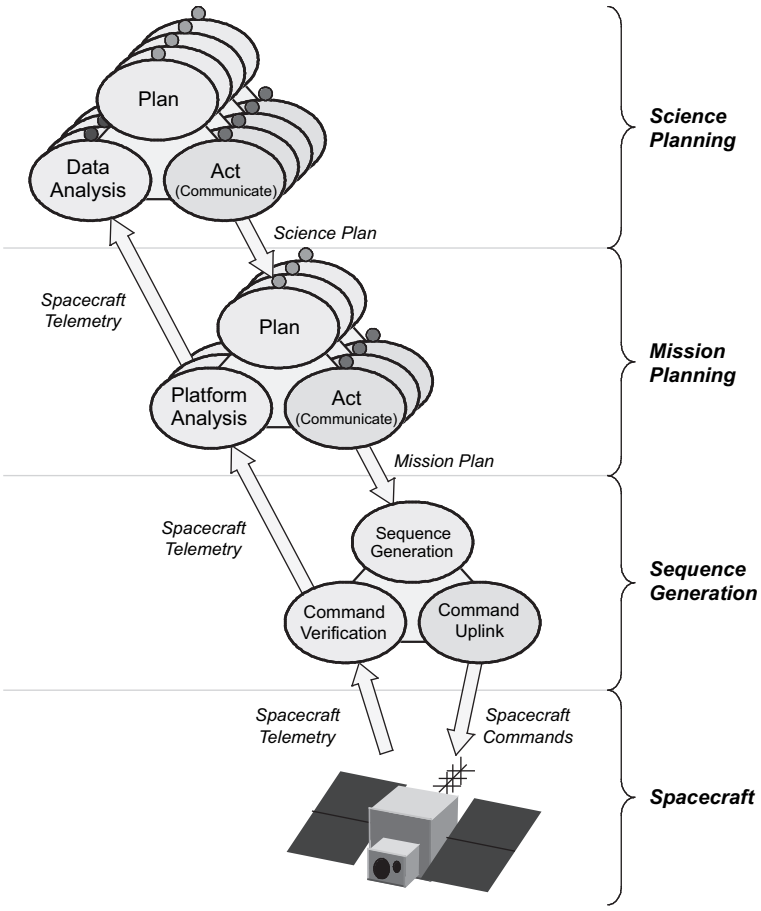

Fig. 7.9. Cooperative autonomy view of spacecraft mission control.

The cooperation found in the mission planning group is similar to that of science planning. One difference is that their goals come from the science planning group instead of being self generated. Another difference is that the mission group augments the science activities with additional activities that must be performed for a successful mission. The product of this group is the mission plan. The sequence planning group receives the mission plan, evaluates the spacecraft telemetry, and adapts its plans accordingly.

The cooperative model merges the sequence planning group and command sequencer components of Figure 7.8 into a single autonomy cycle. The planning element takes the mission plan from the mission planning group and converts it into a series of commands that can be sent to the spacecraft and executed. Once the commands are executed, the spacecraft telemetry reports

the status of the craft, and command verification checks to see whether the uplinked command file functioned properly. If it did not, the science planning, mission planning, and sequence generation must work to recover from the problem. If the sequence planning group is made up of multiple members (the figure only shows one), then they must collaborate in building the plan and in coordinating what will ultimately be sent to the spacecraft.

In the above example, the spacecraft has no internal autonomy. It executes the commands that were uplinked in a file and returns the results. Many spacecraft have a limited amount of internal autonomy as shown in Figure 7.10. This automation is in a hierarchical cooperation relationship with the sequence planning group and is responsible for converting commands into a series of steps, executing them, and monitoring their completion. It also has the important role of making sure the actions commanded by the sequence planning group do not jeopardize the spacecraft or damage its instruments. When such a condition occurs, the spacecraft automatically takes control and places itself into a safe mode. This type of automation is critical to spacecraft that operate very far from earth: the delay in long-range communications, even at the speed of light, means receiving status information, sending commands, and receiving confirmations will take a long time and the spacecraft can be lost before ground control has a chance to react to an unexpected condition. Furthermore, the delay means that the changed conditions at the spacecraft will invalidate the commands based on the earlier conditions, rendering control by the ground ineffective and potentially counterproductive. In short, ground control of a very remote space asset (e.g., a rover or a spacecraft) under dynamic risk conditions is, in general, not an option.

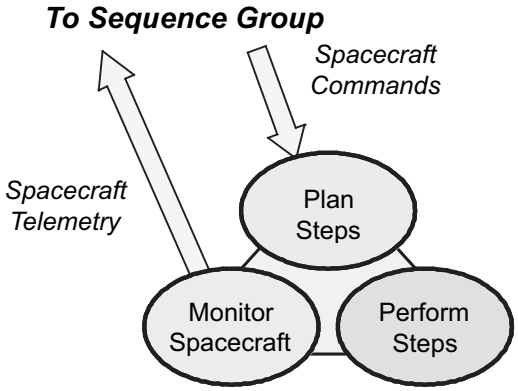

Fig. 7.10. Spacecraft automation.

7.4.2 Analysis of Spacecraft Mission Model

The first thing to note in Figure 7.8 is the limited use of automation technologies. There are large amounts of human cooperation, communication, and negotiation, but the only fully automated processing occurs at the lowest level of the hierarchy. This is where commands are sent to the spacecraft and automatically verified. This lack of automation dictates a large staff of expert personnel in each of the mission domains. While this helps to ensure the safety and success of the mission, it also means substantial operational costs.

The large degree of human communication and negotiation also severely limits the speed at which the organization can perform decision making. The time required for human decision making has three major impacts:

Planning Time: The entire planning hierarchy has developed to accommodate the slowness of human deliberations. Decisions requiring long deliberations are accomplished at the top of the hierarchy and are infrequent. The middle tier of the hierarchy is focused upon near-term decision making and the lowest level on those of the immediate future. If human deliberations can be reduced or eliminated at any of the levels, improvements can be made in the time required for the planning process.

Reaction Time: The cornerstone of the planning hierarchy is predictive scheduling. This requires all possible activities to be pre-planned, with humans involved in all decision making. In the case of a spacecraft anomaly, the mission planning group must be called in to examine the anomaly and re-plan the short term activities. While this re-planning doesn't have a major impact on single platform operations, it can easily lead to non-productive time when one instrument of a platform fails. An automated mission planner could easily redirect the properly functioning instruments to another activity while the anomalous instrument is being examined.

Iteration Time: The long lead time required between science planning and execution on-board the platform restricts the opportunistic science opportunities of a platform. An alteration of the science plan is required for the mission group to refocus the near-term activities.

A great deal of informal communications and negotiations happens between the members of the planning groups. Members use a variety of mechanisms to achieve consensus on the high-level science goals. Frequent meetings, e-mail messages, and telecommunications are used, as well as a variety of planning and scheduling tools. Normally the science planning group produces a document that is passed on to the mission planning group. While this is acceptable when the group is composed of humans, it severely limits the possibilities for automation of group activities. The informal nature of group activities imposes severe limitations on the speed at which the group can reach consensus.

A careful analysis of mission cooperation shows that in some cases the wrong type of human cooperation is being applied to a particular level of the

hierarchy. This is especially clear in the mission planning group. Mission planning is primarily a traditional scheduling problem, dealing with optimizing a candidate list of activities, resources, and constraints. Ideally, an automated scheduling system would perform this task and would focus on maximizing the science output of the mission. The fact that the planning experts are primarily responsible for the spacecraft's safety suggests that optimizing science output will not be their primary focus. Instead, they spend much of their time focusing on spacecraft safety and health issues. While spacecraft health and safety is vitally important, the mission plan should always be optimized for science output while simultaneously guaranteeing that safety and health goals are met.

While these features all impose limitations and constraints on mission effectiveness, the current mission organization has performed reliably for many NASA missions. The cooperative autonomy model does suggest specific areas where improvements can be made and these will be the focus of the next section.

7.4.3 Improvements to Spacecraft Mission Execution

To increase mission science output in the current environment, it is necessary to insert new technologies that decrease the labor needed in building and managing spacecraft. This section will examine some technologies supporting this goal.

The science planning group is responsible for setting goals and interpreting results. It is one area where it is difficult to eliminate the large investment in human labor. However, some technologies can be inserted that will make planning efforts easier and more efficient. Groupware technologies could assist in the planning cycle by making team communication and idea-sharing more efficient. It would also lower the number of face-to-face meetings required by the staff and allow them to work on the project at times convenient to their schedule. In a similar manner, advanced data fusion, analysis, and visualization packages can assist scientists in interpreting their results. Both these technologies are currently in use and their use should be expanded.

Spacecraft sequence generation creates the series of commands necessary to achieve the objective of the high-level mission plan. The process usually engages software tools to support humans. As stated in Section 7.4.1, some spacecraft already have a limited amount of automation in performing a similar function. It would be a reasonably small step to move the sequence generation directly into the spacecraft. This would allow the spacecraft to control its activities and would lower the human staffing requirements.

Between science planning and sequence generation is the mission planning function. This is an area ripe for automation. Mission planning groups already use planning and scheduling software to deal with the more detailed and labor-intensive tasks. By augmenting the existing software, it would be possible to design a system that is completely automated for normal operating

conditions, eventually lowering the need for human labor for mission planning. The mission planning system would initially be run on ground-based systems to facilitate monitoring and problem resolution. Figure 7.11 shows this configuration. While not shown in the figure, a human mission manager would probably monitor the mission planning system and would, if necessary, resolve problems.

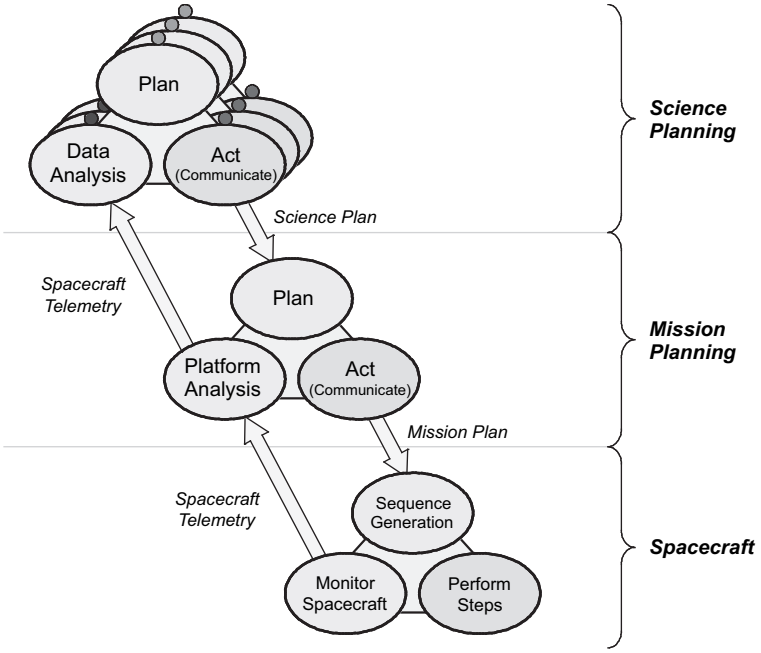

Fig. 7.11. Cooperative Autonomy View of Spacecraft Mission Control.

Eventually, the mission planning function could be installed on the spacecraft, thereby enabling it to operate fully autonomously. Figure 7.12 shows this final system, where human labor is focused on creating a science plan and interpreting the results. The spacecraft converts the science plan into a mission plan and then converts the mission plan into a series of low level commands. Following execution of the low level commands, the results are evaluated and the collected data is sent to the science team for analysis and interpretation. Possibly, the team will modify the plan and redirect the spacecraft. This model does not eliminate human intervention, since a human mission manager would monitor the system and ensure any problems were properly resolved.

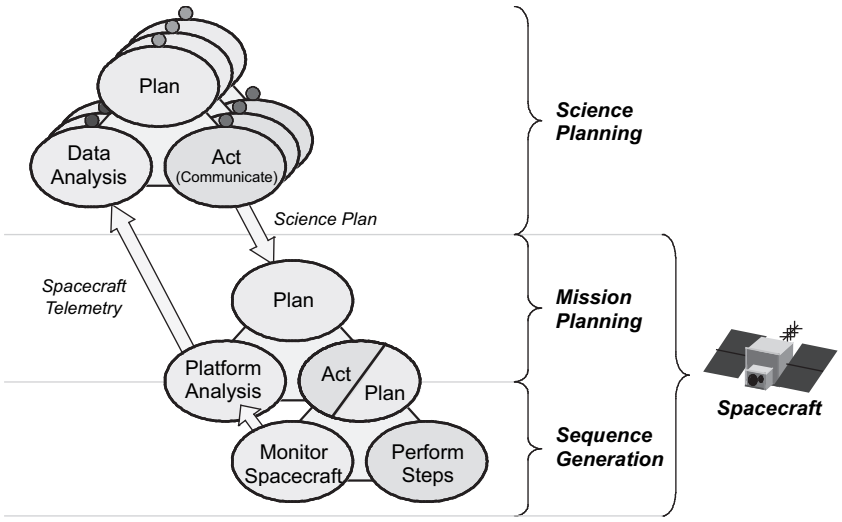

Fig. 7.12. Cooperative Autonomy View of Spacecraft Mission Control.

7.5 An Example of Cooperative Autonomy: Virtual Platform

The previous section discussed the cooperative autonomy model in view of current NASA processes and missions. NASA is also pursuing new ways to increase the science return of spacecraft while minimizing the cost of development and operations. One emerging concept is that of virtual platforms.

Virtual platforms involve the instruments of two or more spacecraft to collect data for a common science goal. Many configurations of virtual platforms are possible. In the simplest example, known as formation flying, multiple spacecraft perform their science collection while keeping a fixed position relative to one another. The fixed relative position can, in some cases, be maintained without direct communication between the spacecraft, and the cooperation in such cases is limited to merging the collected data.

Simple constellations are groups of identical spacecraft that coordinate their data collection. As in formation flying, merging the data collected by the constellation enables a more complete view of the science. Advanced constellations are able to collaborate during planning phases of the mission, which allows them to allocate tasks to the most suitable spacecraft.

Complex constellations are heterogeneous mixes of different spacecraft. They share the characteristics of simple constellations, but differences in spacecraft sensors allow collections in either multiple spectra (e.g., infrared (IR) and ultraviolet (UV)) or different disciplines (e.g., earth radiation and atmospheric composition). These differences make the resulting data fusion

more difficult but allow richer, augmented sets of science data. Further, in such a configuration, older pre-existing spacecraft may be used in new ways not planned by the original designers of the spacecraft.

This section will now use the cooperation models previously developed to highlight issues related to the development of virtual platforms.

7.5.1 Virtual Platforms under Current Environment

For virtual platforms to be effective, mission control must be able to select appropriate spacecraft for data collection and then task them. The current mission management organization, being designed for single platforms, does not scale well when managing multiple platforms. Figure 7.13 shows the mission management structure for a two-spacecraft virtual platform using current management techniques. Since the science planning group sets the goals for the whole virtual platform, the group is shared among all the vehicles of the virtual platform. The group also has the responsibility for fusing the data returning from all spacecraft.

Each vehicle has its own mission planning group. This group is responsible for converting the science plan into a mission plan appropriate for the specific vehicle. This is necessary because each spacecraft will have a different role. It might be possible to share human planners if the cooperating spacecraft were similar and the planning demands were modest.

The sequence generation and monitoring is very specific to each vehicle, because each vehicle has a specific role to play and because sequence generation is focused upon platform-specific issues like the battery charge or damaged instruments. It is therefore unlikely that the human operators in these roles could easily be shared.

Given the current mission management structure, virtual platforms would make serious demands on NASA. Assume, for example, that the platforms being used have a four member science team and require three mission planners and one command sequence operator. Managing one spacecraft would therefore need the efforts of eight team members. Figure 7.13 represents a two-spacecraft virtual platform, which requires twelve team members. The components of Figure 7.13 that are shaded are those that must be replicated to add additional spacecraft to the virtual platform. Therefore, a ten-spacecraft virtual platform would require forty-four team members. Unless the science is of a very high priority, it is unlikely NASA would support a large virtual platform system. Some of these issues can be addressed using automation and this will be examined next.

7.5.2 Virtual Platforms with Advanced Automation

Section 7.4.3 discussed how advanced automation could be used to lower the number of team members necessary to manage a single platform. These same

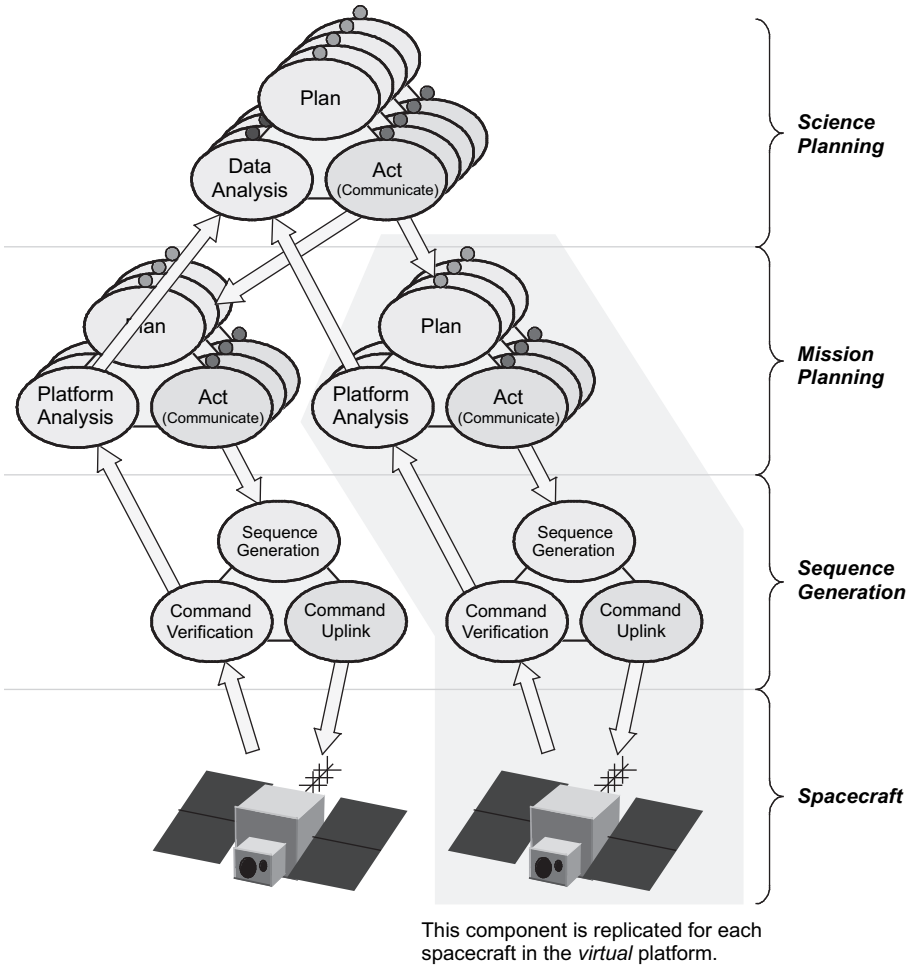

Fig. 7.13. The mission management structure for two spacecraft in a virtual platform configuration.

techniques can be used to create a virtual platform architecture as shown in Figure 7.14.

As in the previous example, the science planning team is shared between all spacecraft that are cooperating as a virtual platform. This is where the similarity ends. Once the science plan is generated, it is communicated directly to the spacecraft. The top level planning component of each spacecraft negotiates with its counterparts on the other spacecraft to determine their

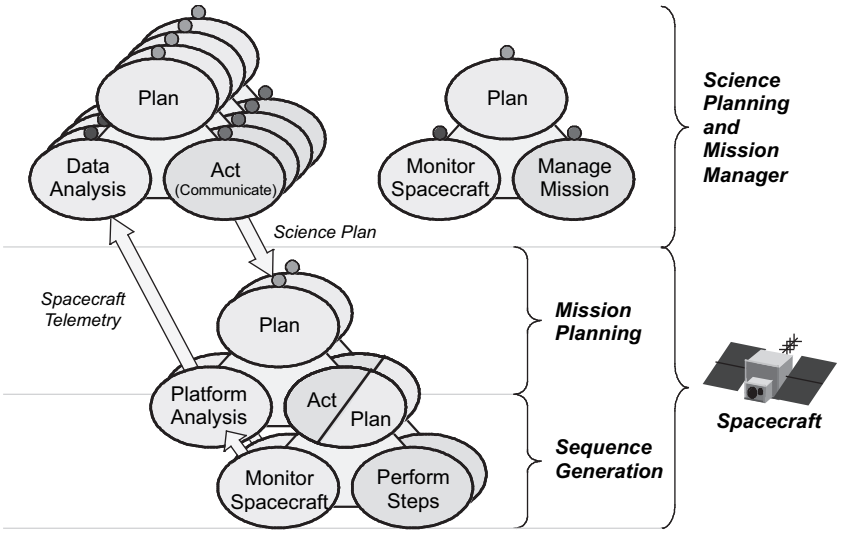

Fig. 7.14. Cooperative Autonomy View of Spacecraft Mission Control.

individual responsibilities in the global mission. Once the negotiations are complete, each spacecraft performs its mission and returns results to the science team. In conjunction with the science team, a small staff will be required to monitor the overall virtual platform and address any problems that may occur.

This approach to virtual platforms is very attractive. Using the numbers from the previous example, this architecture only requires five team members, no matter how many spacecraft are involved in the virtual platform. This compares very favorably to the forty-four staff members necessary to manage a ten-spacecraft virtual platform when the current mission architecture is used.

7.6 Examples of Cooperative Autonomy

Cooperative autonomy requires many different technologies to be synthesized into a functional whole. Aspects of cooperative autonomy can be found in hundreds of projects. This section will outline several projects that incorporate one or more technologies that support the development of cooperative autonomy. The projects were selected to give the reader a cross-section of the technologies available.

New Millennium Program (NMP)

The NMP is a NASA/JPL project that will aggressively demonstrate new technologies for automation and autonomy. Though the primary thrust is technology, the project has scientific goals. NMP will fly a series of deep space and earth-orbiting spacecraft, the first of which launched in 1998. Some of the software technologies are:

- model-based reasoning
- planning and scheduling architectures
- executive architecture (performs plans)
- fuzzy logic
- neural networks

The DS1 spacecraft, launched in 1998, was the first to employ an on-board, autonomous system, AutoNav, to navigate to celestial bodies. About once per week throughout the mission, AutoNav was invoked [114]. The system made navigation decisions about spacecraft trajectory and targeting of celestial bodies with little assistance from ground controllers.

DS1 also used the New Millennium Remote Agent (NMRA) control architecture (Figure 7.15) [101]. In two separate experiments, the remote agent was given control of the DS1 spacecraft. The remote agent involved an on-board mission manager that used a mission plan comprising high-level goals. A planning and scheduling engine generated a set of activities based on the goals, the spacecraft state, and constraints on spacecraft operations. The plan execution component incorporated a hybrid reactive planner and a model-based identification and reconfiguration system. The reactive planner decomposed the activities supplied by the high level planner into primitive activities, which were then executed. The model-based reasoning component used data from sensors to determine the current mode from the current spacecraft state. If a task failed, the model-based component assisted the reactive planner by using its model to act as a recovery expert and determine possible recovery strategies.

This project is highlighted because it is NASA's showcase for new technology, and the software technologies are truly revolutionary. The project will demonstrate substantial autonomy in space-based missions with the goal to establish a virtual presence in space. Cooperative autonomy technologies [22, 39, 43, 109, 194] could augment the DS1 autonomy architecture and further this goal.

7.6.1 The Mobile Robot Laboratory at Georgia Tech

The Georgia Tech Mobile Robot Laboratory (MRL) has been working on the fundamental science and current practices of intelligent mobile robot systems. The MRL has the goal of facilitating the technology transfer of their research results to real world problems.

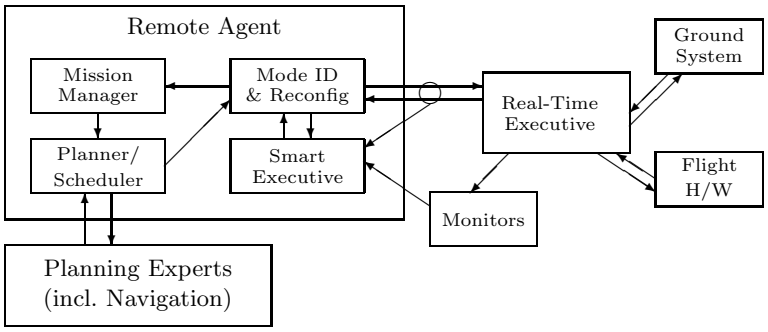

Fig. 7.15. Remote Agent architecture.

Many of the MRL projects should be of interest to those attempting to set up a cooperative autonomy laboratory. The MRL has studied on-line adaptive learning techniques for robotic systems that allow robots to learn while they are actively involved in their operating environment. This type of learning is intended to be fast, similarity-based, and reactive. The MRL has also studied off-line learning where the robot system reasons deeply about its experiences and learns as a result of this analysis. This type of learning is intended to be slower, case-based and explanation-based, deliberative, and goal-oriented.

Georgia Tech has pursued autonomous vehicle research projects supported by the Defense Advanced Research Projects Agency (DARPA). One effort mixed autonomous robot behavior with human controllability. Other research addressed multi-agent systems that achieve tasks in the context of hostile environments.

Georgia Tech has developed several software packages that allow users to create robot control architectures for a specific domain and then test the control architecture in a simulated robot environment. One is written in Java and is designed to be portable. The simulation environment is compatible with off-the-shelf robotic hardware and allows the control architecture developed in the simulator to be run directly on a physical robot.

7.6.2 Cooperative Distributed Problem Solving Research Group at the University of Maine

The University of Maine's Cooperative Distributed Problem Solving Research group is centered on determining and devising the features, characteristics, and capabilities that are sufficient to bring about collaboration among groups of autonomous and semi-autonomous agents toward the accomplishment of tasks. Their work has involved underwater robots, which share many of the same challenges as spacecraft:

- operations in hostile environments

- self-contained operations
- operations in six degrees of freedom (with neutral buoyancy or weightlessness)
- operations with limited communications bandwidth.

The group's research has focused on intelligent control for autonomous systems, cooperative task assignments, determining what has to be communicated during cooperative problem solving, and developing a low-bandwidth conceptual language for cooperative problem solving. In one project, a collection of underwater autonomous vehicles collect data in the ocean and can create a 3-D image of the area of interest. Connected through low bandwidth acoustic modems or radio links, the vehicles coordinate their sampling activities and results reporting.

7.6.3 Knowledge Sharing Effort

The Advanced Research Projects Agency (ARPA), now named the Defense Advanced Research Projects Agency (DARPA), sponsored Knowledge Sharing Effort (KSE). KSE developed methodologies and software for the sharing and reuse of knowledge [107] in support of communication between autonomous systems. KSE was broken up into three separate efforts.

- Knowledge Query and Manipulation Language (KQML) is a language for exchanging information and knowledge between agents [24]. It prescribes a set of performatives that represent different types of agent communication actions (like ask or tell). KQML coordinates the exchange of these performatives between agents.
- Knowledge Interchange Format (KIF) is a formal computer language designed for exchanging complex knowledge between agents [44]. It has declarative semantics and allows the agents to exchange information and describe how that information should be interpreted (the definitions of objects, the meaning of relationships, etc.).
- The final effort is the development of deep knowledge bases for domains of interest. These knowledge bases will have definitions for objects of interest, define relationships and concepts, and populate the knowledge bases with important objects.

KSE is highlighted here because it has been a long-term ARPA/DARPA project to build formal mechanisms for agent communication. In building their languages and systems, the researchers have had to address issues that any group of collaborating agents would have to address.

7.6.4 DIS and HLA

The military has developed a series of high quality training and simulation systems, which immerse the fighting soldier in a detailed model of the area

of combat. The efforts began in the early 80's with SimNet [6, 48], which evolved into Distributed Interactive Simulation (DIS), and then into High Level Architecture (HLA) [18].

These systems create a shared virtual environment where the combat elements (tank, plane, missile, helicopter, etc.) can see themselves and the other combatants. Each soldier sits at a station that controls an element (e.g., a tank position or the cockpit of a fighter) and the soldier's actions cause an appropriate change in the simulation of the element in the virtual world. The soldiers are given a view appropriate to their vehicles and stations and they are able to see the other combatants and the effects of their actions (a missile being fired or a tank turret being rotated). The system has been deemed so good that it has been used to test out new tactics and has been used to assist in the design of new systems by allowing different designs or tactics to be simulated and tested under simulated combat conditions.

Hundreds of individual vehicle simulations can be connected together over a distributed network using specialized protocols running over Internet protocols. These protocols support the efficient exchange of simulation information and allow all participants to experience an appropriate view of the virtual world without requiring an overwhelming amount of computation per station or overloading the network with simulation updates. Work has been directed toward building simulated forces, linking real physical hardware directly to simulated hardware, and building a virtual environment that would allow foot soldiers to engage in simulated combat.

DIS and HLA have been highlighted because they represent the high end of software testing environments for cooperative autonomy. They can support large numbers of simulated objects in a physically distributed environment using Internet protocols. They can also support the integration of real hardware with simulations. If the HLA protocols were modified to meet NASA requirements, the resulting system could allow detailed testing of proposed cooperative autonomy systems, or could allow realistic ground support stations to be integrated into the environment to test new control regimes or to train ground support personnel.

7.6.5 IBM Aglets

Many different technologies have been proposed to support agent-based programming. One system developed by IBM supports the creation of Aglets, which extends Java based applets to create mobile software agents [83]. The Aglet toolkit helps the programmer develop autonomous agents, which can then be instantiated, cloned, moved to other computation systems, or destroyed. Implemented in Java, the Aglets have an advantage that they can run on any computation platform that supports Java, and they automatically have the many security features provided by Java. The Aglet toolkit does not focus on cooperation between Aglets, but these services could be provided

by other Java classes. IBM has made the Aglet toolkit publicly available to support experimentation by others.

Aglets have been discussed as one of the many possible agent architectures. While they have limited services, they are written in Java and therefore are portable and extensible. It would be possible to augment Aglets with, for example, KQML or FIPA-ACL for inter-agent communication along with the robotic control system of Georgia Tech to build cooperating smart agents.

8

Autonomic Systems

NASA requires many of its future missions (spacecraft, rovers, constellations/swarms of spacecraft, etc.) to possess greater capabilities to operate on their own with minimal human intervention or guidance [180, 181, 182]. Autonomy essentially describes independent activity toward goal achievement, but space-system autonomy alone is not sufficient to satisfy the requirement. Autonomicity, the quality that enables a system to handle effects upon its own internal subsystems and their interactions when those effects correspond to risks of damage or impaired function, is the further ingredient of space assets that will become more essential in future advanced space-science and exploration missions. Absent autonomicity, a spacecraft or other asset in a harsh environment will be vulnerable to many environmental effects: without autonomic responses, the spacecraft's performance will degrade, or the spacecraft will be unable to recover from faults. Ensuring that exploration spacecraft have autonomic properties will increase the survivability and therefore their likelihood of success. In short, as missions increasingly incorporate autonomy (self-governing of their own goals), there is a strong case to be made that this needs to be extended to include autonomicity (mission self-management [160]). This chapter describes the emerging autonomic paradigm, related research, and programmatic initiatives, and highlights technology transfer issues.

8.1 Overview of Autonomic Systems

Autonomic Systems, as the name suggests, relates to a metaphor based on biology. The Autonomic Nervous System (ANS) within the body is central to a substantial amount of non-conscious activity. The ANS allows us as individuals to proceed with higher level activities in our daily lives [63] without having to concentrate on such things as heartbeat rate, breathing rate, reflex reactions upon touching a sharp or hot object, and so on [42, 146, 161]. The aim of using this metaphor is to express the vision of something similar to be achieved in computing. This vision is for the creation of the self-management

of a substantial amount of computing functions to relieve users of low level management activities, allowing them to emphasize the higher level concerns of the pursuit of happiness, in general, or the activity of the moment, such as playing in a soccer match, cooking a meal, or engaging in a spirited scientific debate.

The need and justification for Autonomic Systems arises from the ever increasing complexity of modern systems. A not uncommon complaint about the information technology (IT) industry identifies its inordinate emphasis on improving hardware performance with insufficient attention to the burgeoning of software features that always seem to require every possible bit of additional hardware power, to the neglect of other vital criteria. This has created a trillion dollar industry with consumers at the mercy of the hardware-software upgrade cycle. The consequence is a mass of complexity within 'systems of systems', resulting in an increasing financial burden per computer (often measured as the TCO: total cost of ownership).

In addition to the TCO implications, complexity poses a hinderance to achieving dependability [156]. Dependability, a desirable property of all computer-based systems, includes such attributes as reliability, availability, safety, security, survivability and maintainability [8]. Dependability was identified by both US and UK Computer Science Grand Research Challenges: "Build systems you can count on", "Conquer system complexity" and "Dependable systems (build and evolution)" [60]. The autonomic initiatives offer a means to achieve dependability while coping with complexity [156].

8.1.1 What are Autonomic Systems?

An initial reaction to the Autonomic Initiative was "is there anything new?", and to some extent this question can be justified, as Artificial Intelligence (AI), and Fault Tolerant Computing (FTC), among other research disciplines, have been researching many of the envisaged issues within the field of Autonomic Computing for many years. For instance, the desire for automation and effective, robust systems is not new. In fact this may be considered an aspect of best-practice systems and software engineering. Similarly, the desires for systems self-awareness, awareness of the external environment, and the ability to adapt, are also not new, being major goals of several fields within AI research.

What is new is AC's holistic aim of bringing all the relevant areas together to create a change in the industry's direction; selfware instead of the hardware and software feature-upgrade cycle of the past, which created the complexity and TCO quagmire. IBM, upon launching the call to the industry, voiced the state of the industry's concerns as complexity and TCO. They presented the solution to be Autonomic Computing, expressed as comprising eight elements [63]:

- Possess system identity: detailed knowledge of components
- Self configure and re-configure: adaptive algorithms

- Optimize operations: adaptive algorithms
- Recover: no impact on data or delay on processing
- Self protection
- Aware of environment and adapt
- Function in a heterogeneous world
- Hide complexity

These eight elements can be expressed in terms of properties that a system should posses in order to constitute autonomicity [156]. These are described in Section 8.1.2 and elaborated upon in Section 8.1.3, which discusses the very constructs that constitute these properties.

8.1.2 Autonomic Properties

System autonomicity corresponds to the presence of the properties depicted in Figure 8.1 [156]. The general properties of an autonomic (self-managing) system can be summarized by four objectives:

- Self-configuring
- Self-healing
- Self-optimizing
- Self-protecting

which are referred to as self-chop, and four attributes:

- Self-awareness
- Environment-awareness
- Self-monitoring
- Self-adjusting

Essentially, the objectives represent broad system requirements, while the attributes identify basic implementation mechanisms. Since the 2001 launch of Autonomic Computing, the self-* list of properties has grown substantially [169], yet this initial set still represents the general goal.

The self-configuring objective represents a system's ability to readjust itself automatically, either in support of changing circumstances or to assist in self-healing, self-optimization, or self-protection. Self-healing, in reactive mode, is a mechanism concerned with ensuring effective recovery when a fault occurs—identifying the fault and, where possible, recovering from it. In proactive mode, it monitors vital signs and attempts to predict and avoid health problems. Self-optimization means that a system is aware of its ideal performance, can measure its current performance against that ideal, and has policies for attempting improvements. It may also react to policy changes within the system as indicated by the users. A self-protecting system will defend itself from accidental or malicious external attack. This means being aware of potential threats and having ways of handling those threats [156].

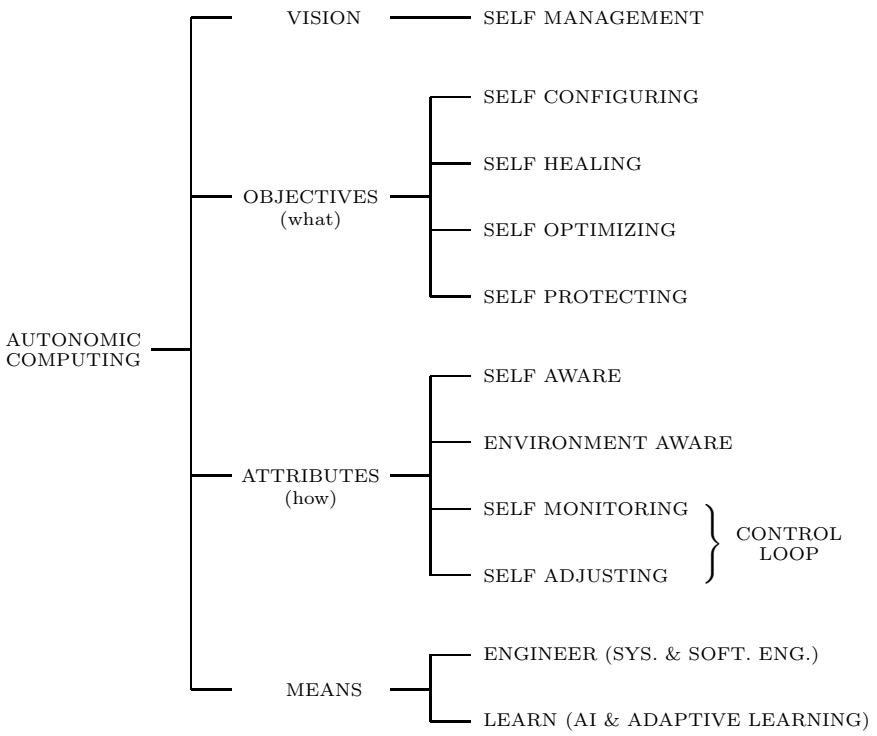

Fig. 8.1. Autonomic Computing Properties Tree.

In achieving self-managing objectives, a system must be aware of its internal state (self-aware) and current external operating conditions (environment-aware). Changing circumstances are detected through self-monitoring, and adaptations are made accordingly (self-adjusting) [156]. Thus, a system must have knowledge of its available resources, its components, their desired performance characteristics, their current status, and the status of inter-connections with other systems, along with rules and policies of how these may be adjusted. In the broad view, the ability to operate in a heterogeneous environment will require the use of open standards to enable global understanding and communication with other systems [63].

These mechanisms are not independent entities. For instance, recovery from a successful attack will include self-healing actions and a mix of self-configuration and self-optimization: self-healing to ensure dependability and continued operation of the system, and self-configuration and self-optimaztion to increase self-protection against similar future attacks. Finally, these self-mechanisms should ensure that there are minimal disruptions to the pursuit of system goals.

There are two main perceived approaches (Figure 8.1) considered to be the means for Autonomic Computing to become a reality [146]:

- Engineer Autonomicity
- Learn Autonomicity

'Engineer Autonomicity' has an implied Systems and/or Software Engineering view, under which autonomic function would be engineered into the individual systems. 'Learn Autonomicity' has an implied AI, evolutionary computing, and adaptive learning view, where the approach would be to utilize algorithms and processes to achieve autonomic behavior. However, both approaches rely on each other in achieving the objectives set out in Automatic Computing. Autonomic Computing may prove to require a greater collaboration between the intelligence-systems research and system- and software-engineering fields to achieve the envisaged level of adaptation and self-management within the Autonomic Computing Initiative.

8.1.3 Necessary Constructs

Considering these autonomic properties, the key constructs and principles that constitute an Autonomic Environment are:

- Selfware; Self-*
- $AE = MC + AM$
- Control Loop; Sensors+Effectors
- $AE \leftrightarrow AE$

Selfware; Self-*: The principle of selfware (self-managing software and firmware) and the need for self-* properties were discussed in the previous sections.

AE=MC+AM: Figure 8.2 represents a view of an architecture for an autonomic element, which consists of the component to be managed and the autonomic manager [69, 154]. It is assumed that an autonomic manager (AM) is responsible for a managed component (MC) within a self-contained autonomic element (AE). This Autonomic Manager may be designed as part of the component or may be provided externally to the component, as an agent, for instance. Interaction will occur with remote autonomic managers (e.g., through an autonomic communications channel) through virtual, peer-to-peer, client-server [11], or grid [33] configurations.

Control Loop, Sensors+Effectors: At the heart of any autonomic system architecture are sensors and effectors [42]. A control loop is created by monitoring behaviour through sensors, comparing this with expectations (historical and current data, rules, and beliefs), planning what action is necessary (if any), and then executing that action through effectors [68]. The control loop, a success of manufacturing science for many years, provides the basic backbone structure for each system component [41].

IBM represents this self-monitor-self-adjuster control loop as the MAPE (Monitor, Analyze, Plan, and Execute) control loop. The monitor and analyze parts of the structure process information from the sensors to provide both self-awareness and an awareness of the external environment. The plan and execute parts decide on the necessary self-management behavior that will be executed through the effectors. The MAPE components use the correlations, rules, beliefs, expectations, histories, and other information known to the autonomic element, or available to it through the knowledge repository within the AM.

AE ↔ AE: The Autonomic Environment requires that autonomic elements, and, in particular, autonomic managers, communicate with one another concerning self-* activities to ensure the robustness of the environment. Figure 8.2 views an AE with the additional concept of a pulse monitor (PBM). This is an extension of the embedded systems heart-beat monitor (HBM), which safeguards vital processes through a regular emitting of an 'I am alive' signal to another process, with the capability to encode health and urgency signals as a pulse [148]. Together with the standard event messages on the autonomic communications channel, this provides not only dynamics within autonomic responses, but also multiple loops of control, such as reflex reactions, among the autonomic managers [159].

8.1.4 Evolution versus Revolution

In recognition of, first, the need for differing levels of human involvement and, second, the reality that the overarching vision of autonomic computing will not be achieved overnight, Autonomic Computing maturity and sophistication have been categorized into five "stages of adoption" [10, 20, 69]: *Basic*, *Managed*, *Predictive*, *Adaptive*, and *Autonomic*.

Assessing where a system resides within these autonomic maturity levels is not necessarily an easy task. Efforts are underway to define the required characteristics and metrics [89]. The overall AC maturity is established from a combination of dimensions forming a natural continuum of autonomic evolution [81], such as increasing functionality (manual, instrument-and-monitor, analysis, closed-loop, to closed-loop-with-business-priorities) and increasing scope (sub-components, single-instances, multiple-instances-same type, multiple-instances-different types, to business-systems) [81]. Since assessment is becoming even more complex, efforts are currently underway to automate the assessment process itself [41, 130]. These efforts imply that the Autonomic Computing Initiative is following an evolutionary path.

8.1.5 Further Reading

The best starting point for further reading is IBM's 'call to arms' launch of the initiative [63], the autonomic 'vision' paper [80], and the 'dawning' paper [42], as well as news about the autonomic initiative [108].

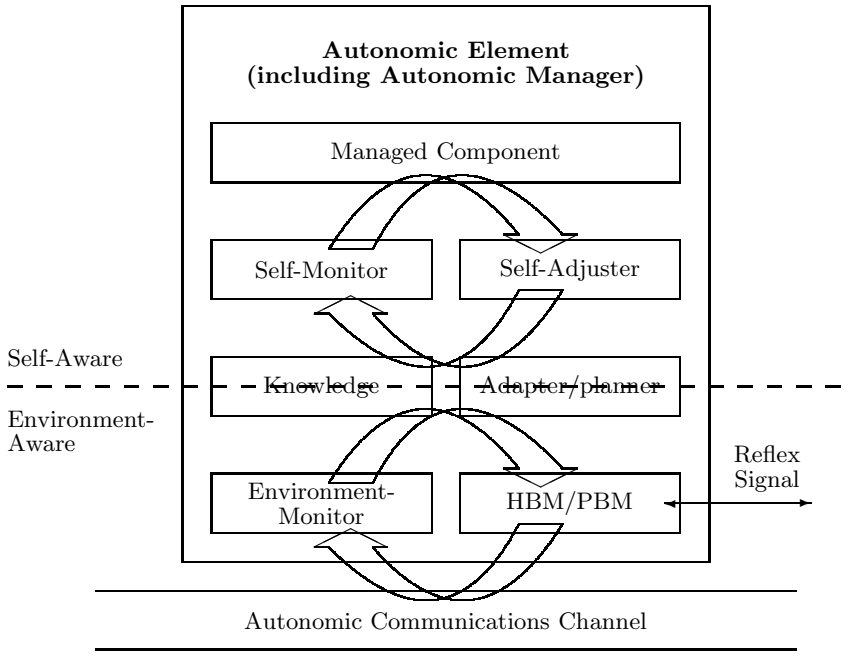

Fig. 8.2. Autonomic Element consisting of Autonomic Manager and Managed Component.

Since the launch of AC, IBM has released various white papers. The general concepts within these have essentially been brought together into a book published by IBM Press [99]. This book covers IBM's view of Autonomicity and how it strategically fits within their other initiatives (such as On-Demand).

Origins of some of the IBM thinking on autonomic computing can be attributed to the Active Middleware Services (AMS) community, where their 5th workshop in Seattle in 2003 became the Autonomic Computing Workshop [105] and evolved, with IBM's backing, into the Autonomic Conference (New York 2004) [72]. The early focus at this stage was very much on its roots, i.e., middleware, infrastructures, and architectures. Other Autonomic workshops include the Workshop on AI for Autonomic Computing, Workshop on Autonomic Computing Principles and Architectures, Workshop on the Engineering of Autonomic Systems, Almaden Institute Symposium: Autonomic Computing, Workshop on Autonomic Computing Systems, and the Autonomic Applications Workshop; and related workshops such as the ACM Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN), and the ACM Workshop on Self-healing Systems (WOSS).

Special issue journals are also beginning to appear [53, 170]. The papers in [53] generally cover engineering topics such as mirroring and replication of servers, software hot swapping, and database query optimization. Those

in [170] strongly represent autonomic efforts for the grid, web, and networks. Appreciating the wider context of autonomic computing, the boiling pot that influenced AC can be found in other research initiatives such as Recovery Oriented Computing [19].

8.2 State of the Art Research

It has been highlighted that meeting the grand challenge of Autonomic Systems will involve researchers in a diverse array of fields, including systems management, distributed computing, networking, operations research, software development, storage, artificial intelligence, and control theory, as well as others [42]. There is not the space here to cover all the excellent research underway, so this section will discuss a selection of the early reports in the literature of state-of-the-art efforts in AC [147].

8.2.1 Machine Design

A paper in [70] discusses affect and machine design [102]. Essentially, it supports those psychologists and AI researchers that hold the view that affect (and emotion) is essential for intelligent behavior [139, 140]. It proposes three levels for the design of systems:

1. Reaction—the lowest level, where no learning occurs but where there is an immediate response from the system to state information coming from sensory systems.
2. Routine—the middle level, where largely routine evaluation and planning behaviors take place. The system receives inputs from sensors as well as from the reaction level and reflection level. At this level of assessment, there are results in three dimensions of affect and emotion values: positive affect, negative affect, and (energetic) arousal.
3. Reflection—the top level of the system receives no sensory input and has no motor output; it receives all inputs from below. Reflection is a meta-process where the mind deliberates about the system itself. Operations at this level look at the system's representations of its experiences, its current behavior, its current environment, etc.

Essentially, the reaction level sits within the engineering domain, monitoring the current state of both the machine and its environment, and produces rapid reaction to changing circumstances. The reflection level may reside within an AI domain utilizing its techniques to consider the behavior of the system and learn new strategies. The routine level may be a cooperative mixture of both the reactive and reflection levels.

8.2.2 Prediction and Optimization

A method known as Clockwork provides predictive autonomicity by regulating behavior in anticipation of need. It involves statistical modeling, tracking, and forecasting methods [127] to predict need and is now being expanded to include real-time model selection techniques to fulfill the self-configuration element of Autonomic Computing [128]. This work includes probabilistic reasoning and, prospectively, should be able to benefit from invoking genetic algorithms for model selection.

Probabilistic techniques such as Bayesian networks (BNs) discussed in [50] are also central in research into autonomic algorithm selection, along with self-training and self-optimizing [50]. Re-optimization of enterprise business objectives [4] can be encompassed by the breadth and scope of the autonomic vision through such far-reaching work combined with AI techniques (machine learning, Tabu search, statistical reasoning, and clustering analysis).

As an example, the application 'Smart Doorplates' assists visitors to a building by locating individuals who are not in their offices. A module in the architecture utilizes probabilistic reasoning to predict the next location of an individual, which is reported along with his/her current location [173, 174].

8.2.3 Knowledge Capture and Representation

Vital to the success of Autonomic Systems is the ability to transfer expert human knowledge about system management and configuration to the software managing the system. Fundamentally, this is a knowledge-acquisition problem [86]. One current research approach is to capture the expert's actions automatically (keystrokes and mouse movements, etc.) when performing on a live system, and dynamically build a procedure model that can execute on a new system and repeat the same task [86]. Establishing a collection of traces over time should allow the approach to develop a generic and adaptive model.

The Tivoli management environment approaches this problem by capturing in its resource model the key characteristics of a managed resource [78]. This approach is being extended to capture the best practices information into the common information model (CIM), through descriptive logics at both the design phase and the deployment phase of the development lifecycle [84]. In effect, the approach captures system knowledge from the creators, ultimately to perform automated reasoning when managing the system.

8.2.4 Monitoring and Root-Cause Analysis

Event correlation, rule development, and root-cause analysis are important functions for an autonomic system [155]. Early versions of tools, and autonomic functionality updates to existing tools and software suites in this area, have recently been released by IBM [41] through their AlphaWorks Autonomic Zone website. Examples include the Log and Trace Tool, the Tivoli Autonomic Monitoring Engine, and the ABLE rules engine.

The generic Log and Trace Tool correlates event logs from legacy systems to identify patterns. These patterns can then be used to facilitate automation or support debugging efforts [41]. The Tivoli Autonomic Monitoring Engine essentially provides server-level correlation of multiple IT systems to assist with root-cause analysis and automated corrective action [41]. The ABLE rules engine can be used for more complex analysis. In effect, it is an agent-building learning environment that includes time series analysis and Bayes classification among others. It correlates events and invokes the necessary action policy [41].

It has been noted that correlation, rule discovery, and root-cause analysis activities can benefit from incorporating Bayesian Networks [153], either in the rule discovery process or in the actual model learning to assist with self-healing [150]. Large-scale server management and control has also received similar treatment. Event logs from a 250-node large-scale server were analyzed by applying a number of machine-learning algorithms and AI techniques to establish time-series methods, rule-based classification, and Bayesian network algorithms for a self-management and control system [129].

Another aspect of monitoring and root-cause analysis is the calculation of costs, in conjunction with the self-healing equation in an autonomic system. One approach utilizes naive Bayes for cost-sensitive classification and a feedback approach based on a Markov decision process for failure remediation [90]. The argument is easily made that the autonomic system involves decisions and decisions involve costs [25]. This naturally leads to work with agents, incentives, costs, and competition for resource allocation and extensions thereof [25, 106].

8.2.5 Legacy Systems and Autonomic Environments

Autonomic Systems is arguably widely believed to be a promising approach to developing new systems. Yet organizations continue to have to deal with the reality of either legacy systems or building 'systems of systems' composed of new and legacy components that involve disparate technologies from numerous vendors [77]. Work is currently underway to add autonomic capabilities to legacy systems in areas such as instant messaging, spam detection, load balancing, and middleware software [77].

Generally, the engineering of autonomic capabilities into legacy systems involves providing an environment for monitoring the system's sensors and providing adjustments through effectors to create a control loop. One such infrastructure is Kinesthetics eXtreme (KX). It runs a lightweight, decentralized, easily integratable collection of active middleware components tied together via a publish-subscribe (content-based messaging) event system [77]. Another tool, called Astrolabe, may be used to automate self-configuration and monitoring and control adaptation [14]. The AutoMate project, incorporating ACCORD (an autonomic component framework) utilizes the Distributed Interactive Object Substrate (DIOS) environment to provide mecha-

nisms to directly enhance traditional computational objects/components with sensors, actuators, rules, a control network, management of distributed sensors and actuators, interrogation, monitoring, and manipulation of components at runtime through a distributed rule-engine [3, 91, 98].

8.2.6 Space Systems

As discussed earlier, with the increasing constraints on resources and the greater focus on the cost of operations, NASA and others have started to utilize adaptive operations and move towards almost total onboard autonomy in certain classes of mission operations [176, 195]. Autonomy provides self-governance, giving responsibility to the agents within the system to meet their defined goals. Autonomicity provides self-management in addition to self-governance as essential to a system's ability to meet its own functional goals. There is also a shared responsibility to ensure the effective management (through self-* properties) of the system, which may include responsibilities beyond the normal task-oriented goals of an individual agent. For instance, monitoring another agent's health signs to ensure self-protection, self-healing and self-configuration and/or self-optimization activities take place as needed. Autonomic Computing, then, can be identified as a key technology [27, 66, 146, 151] for future NASA missions, and research is paving the way for incorporation of both autonomicity and autonomy [182]. These will be discussed in more detail later in Part III of this book.

8.2.7 Agents for Autonomic Systems

Agents, as autonomous entities, have the potential to play a large role in Autonomic Systems [49, 63, 103, 106, 168, 169], although, at this stage, there are no assumptions that agents must necessarily be used in an autonomic architecture. However, as in complex systems, there are substantial arguments for designing a system with agents [75]. Agents can help provide inbuilt redundancy and greater robustness [67], as well as help retrofit legacy systems with autonomic capabilities [77]. With reference to work previously mentioned, a potential contribution of agents may come from environments that require either learning, rules, and norms, or agent monitoring systems.

8.2.8 Policy Based Management

Policy-based management becomes particularly important with the future vision of Autonomic Computing, where a manager may simply specify the business objectives and the system will make it so—in terms of the needed information and communications technology (ICT) [95]. A policy-based management tool may reduce the complexity of product and system management by providing a uniform cross-product policy definition and management infrastructure [41].

8.2.9 Related Initiatives

Other self-managing initiatives include:

- Cisco (Adaptive Network Care) [71],
- HP (Adaptive Infrastructure) [54],
- Intel (Proactive Computing) [187],
- Microsoft (Dynamic Systems Initiative) [96], and
- Sun (N1) [164].

All of these initiatives are concluding that the only viable long-term solution is to create computer systems that manage themselves.

The latest related research initiative is Autonomic Communications [149, 150]. An European Union brainstorming workshop in July 2003 to discuss novel communication paradigms for 2020 identified 'Autonomic Communications' as an important area for future research and development [142]. This can be interpreted as further work on self-organizing networks, but is undoubtedly a reflection of the growing influence of the Autonomic Computing initiative.

Autonomic communications has the same motivators as the autonomic computing concept, except it has a focus on the communications research and development community. Research in autonomic communications pursues an understanding of how an autonomic network element's behaviors are learned, influenced, or changed, and how this effects other elements, groups, and networks. The ability to adapt the behavior of the elements was considered particularly important in relation to drastic changes in the environment, such as technical developments or new economic models [142]. This initiative has now evolved into a major European research program, known as "Situated and Autonomic Communications" (SAC) [141].

8.2.10 Related Paradigms

Related initiatives, as in perceived future computer paradigms, include grid computing, utility computing, pervasive computing, ubiquitous computing, invisible computing, world computing, ambient intelligence, ambient networks, and so on. The driving force behind these future paradigms of computing is the increasing convergence between the following technologies:

- Proliferation of devices,
- Wireless networking, and
- Mobile software.

Weiser first described what has become known as ubiquitous computing [188] as the move away from the "dramatic" machine (where hardware and software was to be so exciting that users would not want to be without it) towards making the machine "invisible" (so embedded in users' lives it would be used without thinking or would be unrecognized as computing). Behind these different terms and research areas lie three key properties:

- Nomadic,
- Embedded, and
- Invisible.

In effect, this may lead to the creation of a single system with (potentially) billions of networked information devices. All of these next generation paradigms, in one form or another, will require an autonomic-self-managing-infrastructure to provide the successful reality of this envisaged level of invisibility and mobility.

Currently, and for the foreseeable future, the majority of users access computing through personal devices. Personal Computing offers unique challenges for self-management in itself due to its multi-device, multi-situation, and multi-user nature. Personal Autonomic Computing is much less about achieving optimum performance or exploiting redundancy (as in AC) and more about simplifying use of the equipment and the associated services [11, 10]. Thus, it is particularly relevant to deal with the move towards a nomadic, embedded, and invisible computing future [152, 157].

8.3 Research and Technology Transfer Issues

The challenge of Autonomic Computing requires more than the re-engineering of today's systems. Autonomic Computing also requires new ideas, new insights, and new approaches.

Some of the key issues that will need to be addressed are:

Trust— Even if the autonomic community manages to 'get the technology right', the trust of the user will be an issue in terms of the user's handing over control to the system. AI and autonomous agent domains have suffered from this problem. For instance, Neural Networks (due to concerns over the 'black-box' approach) and a number of AI techniques (due to their inherent uncertainty) are often not adopted. Rule-based systems, even with all their disadvantages, often win adoption, since the user can trace and understand (and thus implicitly trust) them [153]. Note that even within Autonomic Computing and Autonomic Communications, the bulk of the literature assumes rules will be used instead of other, less brittle and more adaptable stochastic AI approaches.

Economics— new models of reward will need to be designed. Autonomy and Autonomicity may derive another self-* property: selfishness. For instance, why would an autonomic element perform an operation, e.g., relay information, for another AE that was outside its organization and did not affect or benefit from it? In particular, if it was operating within a mobile (battery powered) environment and to do so incurred personal cost, performing the operation for the outside unit could shorten its useful life or make it necessary to recharge earlier.

Standards— The overarching vision of Autonomic Computing will only be achievable through standards, in particular for communicating between AEs. Like the agent community standardizing on a communications protocol, AEs also need a protocol standard so they can be added to a system and be able to communicate immediately. As well, agile ways to define these communications is needed, for which a key enabler would be the self-defining property.

It has been expressed that in ACs initial deployment take-off, many researchers and developers have zeroed in on self-optimization because it is perceived as easier to translate into technology transfer [41]. Essentially, this focus on optimization from the four self-chop attributes may be considered to be going against the grain of technology trends (toward ever faster machines), as such fine-grained optimization is not necessarily a major concern [41]. For Autonomic Computing to succeed in the longer term, the other self-* attributes must be addressed equally and in an integrated fashion.

As well as addressing complexity, Autonomic Computing also offers the promise of a lower total cost of ownership and a reduced maintenance burden as systems become self-managing. Achieving this vision will likely make substantial demands on legacy maintenance budgets in the short-term as autonomic function and behavior is progressively designed into systems.

Achieving the overarching vision of Autonomic Systems will require innovations in systems and software engineering, as well as collaboration involving many other diverse fields. Early R&D presented in this chapter highlights the momentum that is developing on a broad front to meet the vision. The NASA community, with its increasing utilization of autonomy in missions, will only benefit from the evident paradigm shift within computing that brings Autonomicity into the mainstream.

Part III

Applications

9

Autonomy in Spacecraft Constellations

In this chapter, we discuss the application of the autonomy technologies considered in previous chapters to spacecraft constellations. The needs of constellations that can be supported by onboard autonomy are described along with the enhancements attainable by constellation missions through the application of onboard autonomy. A list of hypothetical constellation mission types is also posed and a list of governance concepts is then presented in relation to the degree of central control being exercised on the constellation. Finally, the chapter discusses mobile agent concepts to support autonomic constellations.

9.1 Introduction

As described in Chapter 7, spacecraft constellations are organized into virtual platforms that appear as a single entity to the ground. They are often flown in formation so that different spacecraft can view science phenomena from a different perspective, or view contiguous areas at the same time. Simple constellations are groups of identical spacecraft that coordinate their data collection and merge the collected data to create a more complete view of the science. Complex constellations are heterogeneous mixes of unlike spacecraft. They share the characteristics of simple constellations, but may comprise different types of spacecraft and/or have different instruments. These differences in spacecraft and instruments make the resulting data fusion more difficult but allows richer sets of science data to be collected. This configuration also allows older, pre-existing spacecraft to be used in new ways not thought of by the original designers of the spacecraft [26].

Some examples of NASA constellations are the ST5 and Constellation X missions. The ST5 mission [171], launched in March 2006, has three identical spacecraft that fly in a "string of pearls" formation (Figure 1.1), utilizing a single uplink/downlink to the ground station. Constellation-X (Figure 9.1) involves the use of a small number of telescopes (currently four), in formation

and working together to give the equivalent of a single X-ray telescope for observing black holes and other X-ray sources with greater resolution than before possible [167].

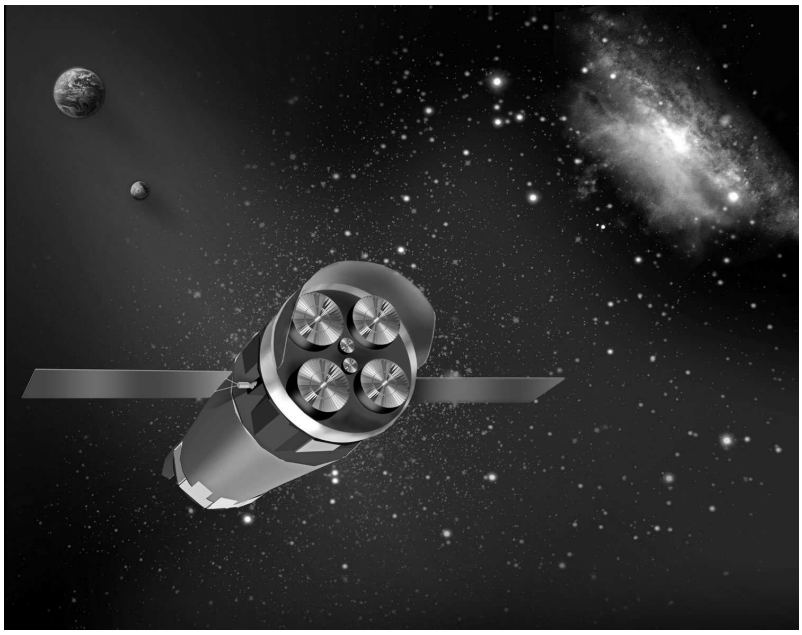

Fig. 9.1. Artist's rendering of a Constellation X X-ray observatory (Image Credit: NASA).

As in other mission types discussed earlier, the motivations for improved autonomy in constellations arise from (among other things) resource constraints pertaining to on-board processor speeds, memory, spacecraft power, etc. Even though on-board computing power will increase in the coming years, the resource constraints associated with space-based processes will continue to be a major factor that needs to be considered when dealing with, for example, agent-based spacecraft autonomy. The realization of "economical intelligence", i.e., constrained computational intelligence that can reside within a process under severe resource constraints (time, power, space, etc.), is a major goal for future missions such as *nano-sat constellations*, where resources are even more constrained due to their small size.

Like other missions, satellite constellation missions can have a wide range of characteristics. Future missions may vary in their data rate and total data volume. Orbits may range from low earth orbits to very elliptical orbits with multi-day periods. Air-to-ground protocols may vary, and the satellites themselves may be low-cost with low autonomy or may be sophisticated with a

high level of on-board self-management. Traditional ground-support systems designed for single satellite support may not efficiently scale up to handle large constellations. The interested reader may refer to [15, 64, 143, 166, 190] for additional information on the challenges of spacecraft constellations.

TYPE	**APPLICATION**	**TYPICAL DESIGN/ MANUFACTURE**	**DATA ACQUISITION**	**OPERATIONS**
Simple (varied number of satellites)	University sponsored. Corporate R&D	Very low cost. Minimally space-rated components.	Not a major issue. Low rate. May operate at amateur radio frequencies.	Extremely low cost. University level.
Cluster: Cluster II (4), Magnetospheric Multiscale (5)	Coordinated science. Virtual telescopes. Stereo imaging.	Complex. Satellite crosslinks. Extensive testing required. High redundancy within satellites.	Not a major issue. Typically high rate due to science mission, but number of satellites is limited or downlink access can be controlled.	Similar to single large satellite. Multiple satellites performing a coordinated function. Added effort for mission.
Coverage-Constellation: Globalstar (48), Orbcomm (36), TIROS (5), NASA NanoSat (100)	Commercial phone/paging/ Internet systems. Earth (or planetary) observation (multi-point data collection, broad survey or coverage).	Satellites operate independently, designed for mass production with limited redundancy, high duty cycle.	Large number of satellites using many ground sites concurrently. Dedicated antenna sites may be needed due to identical satellites working continuously.	May involve hundreds of passes per day. Ideal for automation, as there are many nearly identical passes. Space comm architecture may be need to be fully networked.
Military/ Tactical: XSS-10, ESCORT, Orbital Express	Inspection, imagery.	New concepts are for very small, low-cost, mass-produced spacecraft with no redundancy and minimal mission durations.	Only a few satellites active at a time. May use portable data acquisition sites. May have a video downlink plus minimal status info.	Mostly orbit/ maneuver and data-acquisition activity. Data is for immediate use only. No long-term trending, etc.

Table 9.1. Current and future types of constellation missions and possible issues.

Table 9.1 summarizes current and future types of constellation, their applications, some of the critical distinctions between the applicable mission models, and various relevant issues. To begin to address the implicit challenges, new approaches to autonomy need to be developed for constellations. As discussed in Chapter 4 relative to the ACT prototype, a possible two-step approach for achieving constellation autonomy is as follows:

1. Develop a community of surrogate ground-based agents representing the satellites in the constellation. This will enable the mission to establish, in a prototype environment, the centralized and distributed agent behaviors that eventually will be used in space.
2. Migrate the surrogate agents to the space-based satellites on a gradual or as-needed basis. This step is referred to as *progressive autonomy*.

This chapter will focus on Step 1 and at the end present some ideas relating to Step 2. First we present a brief overview of constellations, reasons for using constellations, and the associated challenges in developing them. These challenges will motivate the agent-based technology discussion in relation to the goal of achieving autonomy in constellations.

9.2 Constellations Overview

Constellations have the potential to provide the data that is needed to yield greater scientific insight and understanding into the cause-and-effect processes that occur in a region. As discussed in Chapter 6, constellations can possess significant, and perhaps obvious, advantages over using just one or two spacecraft. For example, NASA's proposed Magnetotail Constellation (MC) mission, which will use a fleet of 30+ nanospacecraft, would offer space-physics scientists the ability to perform 100 concurrent observations over the magnetosphere, allowing conditions and events recorded to be correlated spatially and temporally.

Constructing and launching constellations will introduce many new and significant challenges. For example, building, launching, and then properly deploying as many as 100 spacecraft housed on one launch vehicle into their required orbits will require the development and demonstration of new spacecraft control solutions so that mission operations costs associated with supporting a constellation comprising a large number of spacecraft do not spiral out of control.

There are a number of implementation issues that are unique to spacecraft constellations. Four examples presented below provide some insight into the challenges that will undoubtedly confront aerospace hardware and software engineers in launching, deploying, and then routinely operating constellations:

- Monitoring engineering telemetry data from one spacecraft is a routine task for mission operations personnel and the ground system computer hardware and software systems. However, responding to time-critical

events and identifying, evaluating, and quickly resolving spacecraft subsystem anomalies can frequently be challenging for humans and computers alike. Effectively monitoring and reacting to conditions reported by the telemetry data from 100 identical spacecraft without also incurring a concomitant and potentially significant increase in staff and ground equipment will be a major challenge.

- Spacecraft that compose a constellation still will need to communicate with the ground system. Commands must be uplinked to the spacecraft, engineering health and safety telemetry data must be transmitted to the missions operations center, and payload data must be returned to the science community for ground-based processing and product distribution. Available (and limited) ground resources (e.g., spacecraft tracking stations, communications networks, and computing resources) will need to be scheduled and managed so that realistic contact plans can be created to support forward and return link telemetry processing for constellations with large numbers of spacecraft. Potentially, advanced space networking technologies will lead to more efficient networked communications capabilities, partially offsetting the need for many direct-to-spacecraft communications paths from ground antennas.
- Some constellation missions may require that the spacecraft communicate with one another for science or formation purposes. One spacecraft may need to broadcast information to many other spacecraft in its vicinity. Alternatively, one spacecraft may need to communicate with another spacecraft in the constellation, for example to cue it to so that the second spacecraft can record an event that the first could not. But these spacecraft may be located in orbital planes where they are rarely if ever in direct line of sight of each other. Multi-hop networked inter-satellite communications may be necessary to synchronize operations of the entire constellation, or just a subset of it.
- Trend analysis is an important element for any spacecraft mission. It helps the mission operations staff determine whether a failure may be imminent so that switchover to backup or redundant subsystems can be performed or, if necessary, to have the spacecraft enter safemode until the problem can be identified and a corrective course of action implemented. Greater automation in ground data processing will be required to support this. Perhaps data mining techniques that are presently implemented for terrestrial databases and e-commerce applications may provide solutions for consideration and adaptation to this new problem.

9.3 Advantages of Constellations

A wide variety of missions could be best implemented with constellations of satellites working together to meet a single objective. Reasons cited for using constellations include lower mission costs, the need for coordinated science,

special coverage or survey requirements, and the need for quick-reaction tactical placement of multiple satellites. The following discusses these in more detail.

9.3.1 Cost Savings

The cost of producing spacecraft for a constellation and getting them to orbit may actually be lower than traditional "one of a kind" satellites that use a dedicated launch vehicle. With a traditional satellite, system reliability requirements force a high level of component protections and redundancy, which leads to higher overall weight and launch costs. Due to their size and/or weight, a dedicated launch is often required for these missions. With a constellation, system reliability can be met by having spare satellites. The use of per-satellite redundancy can be significantly reduced. In some cases, it may be practical to use lower-rated components at a much lower cost combined with an on-orbit sparing plan. Additional savings could be obtained through the use of assembly-line production techniques and coordinated test plans so that the satellites could basically be mass-produced. With a reduced size and weight, new options would be available for launches: lower cost launch vehicles, multiple satellites of the constellation launched on one launch vehicle, and piggy-back launch slots where launch costs are shared with another mission.

9.3.2 Coordinated Science

A constellation of as few as two satellites could be used to perform coordinated science. For example:

- Storms and other phenomena observed from multiple angles could be used to generate 3-D views,
- Satellites with a wide spatial separation could be used for parallax studies of distant objects,
- A cluster of satellites flying in formation and working together could form a virtual lens (or mirror) hundreds of miles across to achieve unprecedented resolution for observations of astronomical objects.

The currently predominant application of satellite constellations aims to extend area coverage. Low earth orbiting constellations such as Globalstar use dozens of satellites to provide continuous global or near-global coverage. The GPS system uses a constellation to provide global coverage and spatial diversity of the multiple satellites in view. Earth imaging missions can use multiple satellites to shorten the time between successive observations of the same area, and can coordinate observations so that dynamic phenomena (hurricanes, earthquakes, volcanic eruptions, etc.) receive augmented attention by additional members of the constellation.

Military applications for constellations include earth observation, weather, and equipment resource monitoring. In the future, it may be possible to launch very small satellites with a very specific purpose and a very short mission duration. The satellites could be produced by the hundreds and launched as needed. The "constellation", at any point in time, would include those satellites currently performing their intended function.

9.4 Applying Autonomy and Autonomicity to Constellations

With the above discussion as motivation, the following section describes how autonomy could be applied in constellation ground control systems and in constellation spacecraft themselves to overcome the issues mentioned above. Finally, the goal of achieving autonomicity in constellations is discussed.

9.4.1 Ground-based Constellation Autonomy

Figure 4.7 (Section 4.3.4, page 89) shows a high-level representation of a constellation simulation of ground-based autonomy for a constellation of four satellites. In this simulation, a number of agents are connected to an environment in which the ground control systems and satellites [177, 183] are simulated. In the simulation, the satellites are in orbit collecting magnetosphere data. The simulation environment propagates the orbits based on ideal conditions. Faults can also be inserted into the telemetry stream to simulate an anomaly.

The group of surrogate spacecraft agents, as a major component of the ground-based community, maintains an awareness of the actual physical constellation. The surrogates act on behalf of their respective spacecraft in status monitoring, fault detection and correction, distributed planning and scheduling, and spacecraft cooperative behaviors (as needed).

A next phase in the evolution of the above ACT scenario will be to have communities of agents each associated with a particular spacecraft in the constellation. Each of these communities would have specialist subsystem agents that would monitor the various subsystems of the spacecraft and cooperate with one another in the handling of anomalous situations. An overall coordinator, or spacecraft agent, would lead the community and represent the spacecraft to ground controllers. It would also represent the spacecraft to other spacecraft agents in the constellation community for activities such as distributed planning and scheduling, and other forms of collaboration.

In the context of spacecraft constellations, the ground-based group of surrogate agents illustrates two major themes in our discussion: (1) surrogate agents can indeed support the concept of constellation autonomy in a meaningful way, and (2) having a ground-based community of surrogate agents

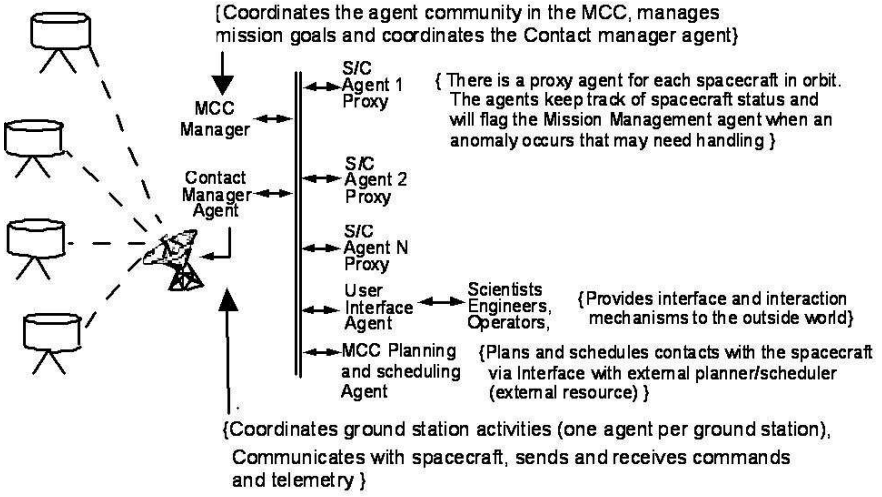

Fig. 9.2. The ACT Testbed Consists of a community of cooperating agents each of which is component-based (from Chapter 4.)

allows developers and users (controllers) to gain confidence and trust in the approach.

9.4.2 Space-based Autonomy for Constellations

Constellation autonomy (as opposed to a single spacecraft autonomy) corresponds to an intrinsic property of the group. Constellation autonomy would not apply to a fleet of spacecraft where each is operated without reference to the others (e.g., a TDRS servicing the communications needs of many user spacecraft). However, if each satellite manages itself in the common external environment so as to maintain a group functionality relative to the environment, even when the members of the group do not communicate directly with one another, and the group accomplishes the end purpose of the constellation, then the constellation can be considered as a self-managing one, i.e., autonomous. This is considered justifiable, from the mission perspective, because the group accomplishes the end result as a system even though its members do not intentionally interchange information. We make this distinction because of the question of autonomy and viability. As long as the constellation produces, i.e., delivers its end purpose on its own without any external support, it is viable and autonomous. Some members of the group may fail, but the remainder will continue to produce, thus maintaining the essential nature of the constellation.

The second and final step (*progressive autonomy*) in the proposed overall plan to realize space-based autonomy is to migrate the spacecraft surrogate

community of agents to the actual spacecraft. As discussed in previous chapters, this is a non-trivial step. A major step in the direction of actual onboard spacecraft autonomy is to have the agent community demonstrate its correctness in actual ground-based spacecraft control centers [178, 184]. This is discussed in Section 9.6.

There are many issues that need to be addressed before this becomes a reality. Some of the major issues are:

- **Adaptation to resource constraints.** As an example, a spacecraft subsystem agent must be able to exist and operate within the microprocessor associated with the subsystem. This is where the concept, which we call "economical intelligence", comes into play. Reasoning code and knowledge and information structures and management need to be "optimized" in order to function properly in the resource-constrained environment of a spacecraft subsystem microprocessor.
- **Integration with existing subsystem autonomy.** As discussed earlier, most spacecraft subsystems already have a degree of autonomy built into their operations. This is realized usually through the use of expert systems or state-based technologies. A subsystem agent should be able to take advantage of the existing capability and build upon it. The existing capability would become an external resource to the subsystem agent that would be used to realize a higher level of autonomy for the subsystem. The agent would need to know about the external resource and how to use it, i.e., factor its information into its reasoning process.
- **Real-time activity.** Most situations experienced by a spacecraft require real-time attention. If the situation is not readily handled by built-in subsystem autonomy, the associated subsystem agent will need to respond in real-time. This will require the agent to have a working reflexive behavior.

9.4.3 Autonomicity in Constellations

A step beyond an autonomous constellation is an autonomic constellation—a collective of autonomous agents that are self-governing and learn individual sequences of actions so that the resultant sequence of joint actions achieves a predetermined global objective. As discussed earlier, this approach is particularly useful when centralized human control is either impossible or impractical, such as whenever timely and adequate communications with humans is impractical. Constellations controlled by *in-situ* "intelligent" spacecraft, in comparison with the more common externally controlled constellation, can have the following advantages:

1. In locations not reachable in a timely manner through human contact:
   a) Initiating and/or changing orders automatically

   b) Evaluating and summarizing global health status, and therefore being able to assess the constellation's ability to conduct a specific exercise, and, further, being able to engage in self-protection and self-reconfiguration, among other self-* functions to ensure the survival and viability of the entire constellation.
   c) Providing summary status of the entire constellation

2. On-the-job training or programming:
   a) Provoking a new mode of behavior on the part of constituent satellites by observing operators and the environment [185].

The role of autonomicity in constellations depends on the needs of the satellites individually and collectively, and on the needs of mission control. For example, how robust an autonomic function on any single satellite must be would depend on such things as the proximity to standard tracking and telecommunications facilities, the urgency of data and command access by the ground, and, for survey missions, the area of simultaneous coverage needed by satellites in the constellation. Availability of communication channels is another factor, whether it be the timing/accessibility and individual channel bandwidth capacity or, in close formation flying, frequency separation of channels. Of course, there are various ways of getting around mutual interference constraints, such as limiting the individual contact events to a single communications link at a time when in close formation, or by compressing the bandwidth requirements, as was mentioned previously; a fully networked inter-spacecraft communications architecture involving multi-hop packet routing also represents an alternative approach for some types of mission. But providing an autonomic function that can address the constraints of a system given the current situation in a mission provides a much more flexible and reusable solution than can be solved by single-point solutions.

The autonomicity of constellation governance influences the spacecraft inter-connectivity design and ground-connectivity design (i.e., human control) in much the same ways as does the individual satellite's subsystem control structure. However, the geometry, scale, and desired performance of the constellation as an integral entity comes into play adding additional design complexity. As a basic consideration, the level of interdependency among constellation members is a factor influenced strongly by mission class, e.g., what type of payload is being carried.

In certain types of mission (for surveillance, analysis, monitoring, etc.), the requirements for accuracy and speed of data or event notification are becoming more demanding. In addition, the resolution requirements for imaging systems require ever larger apertures and hence larger instrument sizes.

Formation flying concepts have been identified as the canonical means for achieving very large apertures in the space environment. (It should be noted that, as with other types of constellation mission, formation flying presents an opportunity to create a synergistic system (e.g., an imaging system) where the

members of the group operate cooperatively to give rise to group capabilities that no single platform could provide by itself. Further, as with other types of constellation mission, the mission can be designed, in many cases, so that the loss of a member leaves the remainder of the group functioning. In the alternative mission design, based on using a single large spacecraft, the loss of the spacecraft means losing the entire mission.)

In a representative formation-flying concept for an astronomical observatory, the formation itself creates the effect of a large "instrument" whose aperture can be changed along with range to target by maneuvering the formation—which presents several issues in control:

- timing,
- timing knowledge accuracy and synchronization,
- positioning, and
- positioning and timing knowledge confidence

and these in turn raise issues of:

- performing inter-satellite communications,
- relaying data, and
- commanding via a master control.

Such control issues generally have no possible solution apart from an autonomous mechanism (e.g., laser cross-links between the members of the formation to permit minute, near-instantaneous relative position adjustments on a scale measured by the diameter of an atomic nucleus), and, for similar reasons (inadequate ability of humans to deal with distant or rapidly occurring phenomena), some level of autonomicity will be required for the system's survival and viability when, for example, the system experiences the effects of a threat that was not predicted.

Autonomic satellite designs will make major contributions to resolving these issues. Each design should be approached first from the viewpoint of constellation architecture, considering the control options available.

9.5 Intelligent Agents in Space Constellations

For single-agent systems in domains similar to space, intelligent machine learning methods (e.g., reinforcement learners) have been successfully used and could be used for single-spacecraft missions. However, applying such solutions directly to multi-agent systems often proves to be problematic. Agents may work at cross-purposes, or have difficulty in evaluating their contribution to achievement of the global objective, or both. Constellations based on intelligent multi-agent systems would have similar challenges. Concepts from collective intelligence [144] could be applied to the design of the goals for the agents so that they are "aligned" with the global goals, and are "learnable"

in that agents could see how their behavior affects their utility and could overcome unforeseen issues.

Satellite intelligence was considered initially by Schetter, Campbell and Surka [133]. They performed comparisons on several high-level agent organizations, along with varying degrees of satellite intelligence, to assess analytically their impact on communication, computation, performance, and reliability. The results indicate that an autonomous, agent-based design provides increased reliability and performance over traditional satellite operations for the control of constellations.

The following sections discuss some of the approaches to using intelligent multi-agents for constellation control.

9.5.1 Levels of Intelligence in Spacecraft Agents

Based on the sum of spacecraft functions, four levels of spacecraft intelligence have been identified [133], where I1 denotes the highest level of intelligence and I4 the lowest level (Figure 9.3) :

- The spacecraft-level agent I4 represents the most "unintelligent" agent. It can only receive commands and tasks from other spacecraft-level agents in the organization, or from the ground, and execute them. An example includes receiving and executing a control command sequence to move to a new position within the cluster. This type of intelligence is similar to that being flown on most spacecraft today.
- The next higher spacecraft-level agent is I3, with local planning functionalities onboard. "Local" means the spacecraft-level agent is capable of generating and executing only plans related to its own tasks. An example would be trajectory planning for orbital maneuvers.
- Agent I2 adds a capability to interact with other spacecraft-level agents in the organization. This usually requires the agent to have at least partial knowledge of the full agent-based organization, i.e., of other spacecraft-level agents. It must therefore continuously keep and update (or receive) an internal representation of the agent-based organization. An example includes coordinating/negotiating with other spacecraft-level agents in case of conflicting requirements.
- The spacecraft-level agent I1 represents the most "intelligent" agent. The primary difference between I1 and the other spacecraft-level agents outlined is that it is capable of monitoring all spacecraft-level agents in the organization and planning for the organization as a whole. This requires planning capabilities on the cluster level (a cluster being a subset of a constellation), as well as a capability by which an agent has full knowledge of all other spacecraft-level agents in the constellation. An example includes calculation of a new cluster configuration and assigning new satellite positions within the cluster.

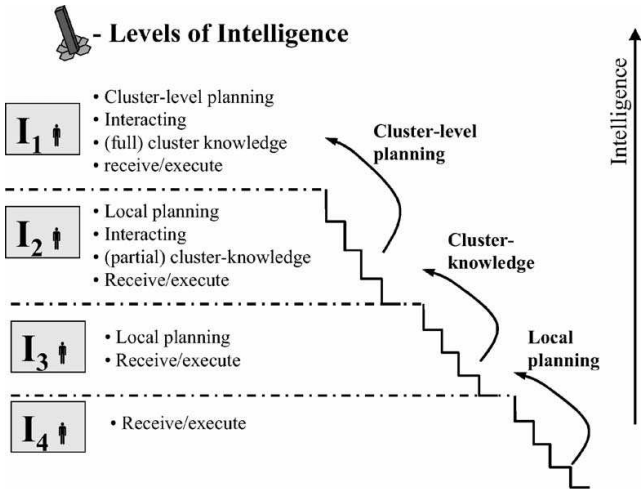

Fig. 9.3. Identification of spacecraft-level agents based on levels of capable intelligence[2].

Selecting the level of intelligence of a multi-agent organization is a complex design process. The organization must be:

- adaptive, able to avoid bottlenecks, and able to reconfigure
- efficient in terms of time, resources, information exchange and processing, and
- distributed in terms of intelligence, capabilities, and resources.

A design selection process starts from an initial spacecraft-level intelligence hierarchy. An example used for TechSat21 [133] is shown in Figure 9.4. Here, high-level mission tasks were decomposed into lower-level tasks. The spacecraft functions required to support these tasks are listed down the left hand column with sub functions grouped by category. Across the top are high level spacecdraft tasks with subtasks underneath. Tasks are then arranged in a matrix form to provide a visualization of the agent capabilities associated with each level of spacecraft intelligence as presented in Figure 9.3. The boxes contain the ID's for function and sub-function categories.

[2]Reprinted from Artificial Intelligence, 145(1-2), Thomas Schetter, Mark Campbell and Derek Surka, Multiple agent-based autonomy for satellite constellations, page 164, Copyright (2003), with permission from Elsevier

[3]Reprinted from Artificial Intelligence, 145(1-2), Thomas Schetter, Mark Campbell and Derek Surka, Multiple agent-based autonomy for satellite constellations, page 154, Copyright (2003), with permission from Elsevier

HIGH-LEVEL TASKS	HT 1: Science Imaging	HT 2: Science Formation Maintaining and Control			HT 3: Science Cluster Reconfiguration		HT 4: Science Cluster Upgrade	
SUB-LEVEL TASKS	ST 11: Science	ST 21: Reject. Disturb.	ST 22: Collis. Avoid.	ST 23: Orbit Maneuv.	ST 31: Fault Detect.	ST 32: Form. Change	ST 41: Accept. new S/C	ST 32: Form. Change
TASK CATEGORY								
INTERACTION 1. Sensing s/c info 2. Transmitting s/c info	F11 F12	F11 F12	F11 F12	F11 F12	F11 F12		F11 F12	
2. DECISION-MAKING 0. Imaging 1. Disturbance 2. Collision Avoidance 3. Failure/Loss 4. Upgrade/Gain	F20	F21	F22		F23		F24	
3. ORGANIZATIONAL 0. Scheduler 1. Planner (Cornwell) 2. Planner (Assign positions) 3. Task allocator 4. FF planner (Trajectory)	F30	F30	F30	F30 F34		F31 F32 F33		F31 F32 F33
4. REPRESENTATIONAL 0. Storing Cluster Information						F40		F40
5. OPERATIVE 0. DAR-Imaging 1. Orbit maneuvering (LQR, bang-bang, FF-command)	F50	F51	F51	F51				

Fig. 9.4. Functional breakdown of the task structure specifically for Techsat21[3].

9.5.2 Multi-Agent Based Organizations for Satellites

Figure 9.5 shows a summary of options as a function of individual spacecraft-level agent intelligence. Note that lower level functional agents are implied in each of the architectures. As can be seen, the number and composition of the different spacecraft-level agents I1–I4 determine the organizational architecture. The top-down coordination architecture includes only one single (highly intelligent) I1 spacecraft-level agent, and the other spacecraft are (unintelligent) I4 agents. The centralized coordination architecture requires at least local planning and possibly interaction capabilities between spacecraft, requiring I3 or I2 agents. The distributed coordination architecture consists of several parallel hierarchical decision-making structures, each of which is "commanded" by an I1 intelligent spacecraft-level agent. In the case of a fully distributed coordination architecture, each spacecraft in the organization represents an I1 spacecraft-level agent, resulting in a totally "flat organization".

[4]Reprinted from Artificial Intelligence, 145(1-2), Thomas Schetter, Mark Campbell and Derek Surka, Multiple agent-based autonomy for satellite constellations, page 166, Copyright (2003), with permission from Elsevier

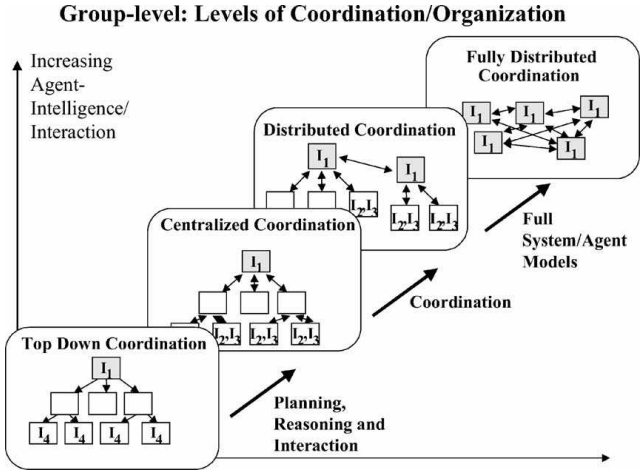

Fig. 9.5. Coordination architectures for coordination of multiple spacecraft-level agents[4].

9.6 Grand View

The next level of space-based autonomy is to develop and verify agents and agent-communities concepts to the point that they can migrate to actual ground-based operations and, when fully verified and validated in an operational context, then migrate to the spacecraft to provide on-board autonomy. Figure 9.6 is a view, a grand view (not the only one), of what such a system might look like [178, 179, 184], and is one possible representation of *progressive autonomy*. It paints a picture in which we can see many threads of agent-based activity, both ground-based and space-based. The major theme of the figure is that of agent migration from one level to another. The figure depicts the various migration paths that could be taken by agents and communities of agents enroute to a spacecraft. This is the essential theme of our proposed approach to realizing complete autonomy for constellations as well as other mission types.

Progressive autonomy refers to the levels of autonomy that can be incrementally achieved by a dynamic community of agents. Achieving a higher level of autonomy in a community means either increasing an already existing agent's capabilities through reprogramming it, introducing a new agent into the community with the desired capabilities, or allowing an agent to develop a new or modified capability via learning.

Progressive autonomy is advantageous for at least two reasons:

1. It allows a new capability to appear from a community of agents supporting an operational mission after that capability has been verified and is trusted outside the testing environment.

2. A qualified agent can be dispatched to a community in need on a temporary basis. Once the need has been fulfilled, the agent can be removed. This keeps the operational resource requirements for the community to a minimum.

Figure 9.6 illustrates some of the concepts that are associated with progressive autonomy in agent-based communities on the ground and in space. There are three levels represented in the figure:

1. An agent development component,
2. A ground-based autonomy component, and
3. A space-based autonomy component.

Agents can migrate from one level to the next depending on their degree of development and validation. Communication between agents on the different levels facilitates the development and validation of agents, since the agents can receive real data from the other levels. The following subsections discuss each of these parts in more detail.

9.6.1 Agent development

The lower part of Figure 9.6 represents the agent-development level, where developers write the code, modify existing agents, or use an automated agent-development system that searches for previously developed agents that perform a needed task and updates them based on new requirements input from the developer.

Domain-specialist agents are available to assist in the development of new agents by interacting with the agents being developed, giving a new agent other agents to interact with and then testing for proper functionality. In addition, data may be received from operational agents in the ground control system and spacecraft for additional testing purposes. Agents in the operational mode would know that messages from agents under development are not operational by virtue of a marking of the messages by the messaging service that passes messages to the operational agents. As agents are developed, they are added to the community of domain experts and provide the developers with additional example agents to modify and test against.

The agent-development level also represents an agent incubator. After an agent is developed, there is an incubation period during which agents are tested in a background or shadow mode. When confidence in the agent's behavior is attained, it is moved into an online community doing real work in its domain. It is at this level that the credentials of the agent come into play. These credentials attest to the development methodology and the verification and validation procedures that would directly ensure the agent's correct behaviors.

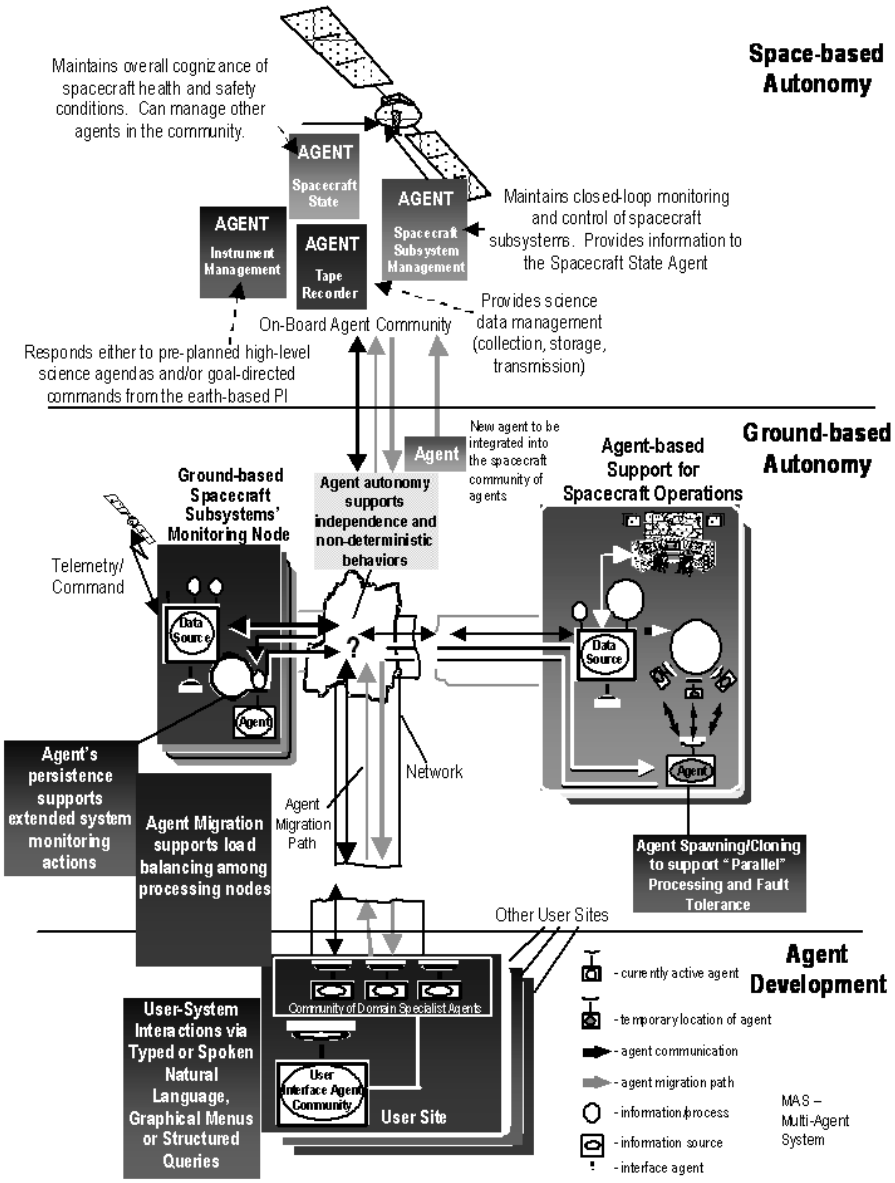

Fig. 9.6. Progressive autonomy of a spacecraft

9.6.2 Ground-based autonomy

The middle section of Figure 9.6 comprises two parts. The right side represents the ground control system, to which some of the agents may migrate after development. In this part, the agent can run in a shadow or background mode where its activities can be observed before it is put into full operation. This allows the users to gain confidence in the agent's autonomy before committing to or deploying it. If a problem is found with the operation of the agent or its operation is not as envisioned or required by the end user, or if enhancements are desired, the agent (or copy of the agent) can be migrated back down to the development area for further modifications or testing. Once modifications are made, it can then be sent back up to the ground operations level, and the process is repeated.

The right side of the middle section also contains an area where agents can be cloned to support parallel processing or fault tolerance: identical agents can be run on multiple (even geographically distributed) platforms.

The left side of the middle section represents the agents that are ready to be sent up to a spacecraft or are operating in a proxy mode. Those agents that are waiting to be sent to a spacecraft may be ones waiting to be uploaded for emergency-resolution purposes (e.g., for anomaly situations), or may carry functionality updates for other agents and are waiting to be uploaded when needed or when resources become available. The agents that are operating in a proxy mode are operating as if they were on the spacecraft, but due (for example) to resource restrictions are temporarily or permanently operating within the ground control system. The proxy agents communicate with other agents and components on the spacecraft as if they were running onboard (subject, as always, to communications constraints).

In a situation requiring an agent to be uploaded, a managing agent in mission control would make a request to the agent-development area (or the repository of validated agents) for an agent with the needed capability, and would (subject to communications constraints) notify the original agent that requested the capability as to the availability of the requested agent. The original requesting agent would then factor this information into its planning, which would be particularly important if the situation were time-critical and alternate actions needed to be planned if the new agent could not be put into service in the constellation within a needed time frame (e.g., as a result of communications vagaries).

9.6.3 Space-based autonomy

The upper part of Figure 9.6 depicts other communities that are purely operational on a spacecraft or other robotic system, the members of which would be mature agents that would have been approved through an appropriate process. These agent communities may be based around spacecraft subsystem (e.g., instrument, recorder, spacecraft state and management, etc.) or

represent a functionality (e.g., anomaly detection, health and safety, science opportunity, etc.).

An agent would be able to migrate from its initial community to other nodes in "agent space" (for lack of a better name). These communities may be logically or physically distinct from the agent's initial community. A single agent may either migrate to a new community when it is no longer needed or clone itself when needed in multiple communities simultaneously.

The idea of realizing constellation autonomy first through ground-based communities of spacecraft surrogate agents and then migrating the agent community to the actual spacecraft is a flexible, dynamic approach to providing ongoing updates to spacecraft functionality. The progressive autonomy that could be realized through this approach would enable mission control to upload only those agents in the community that have been thoroughly verified and in which there is the appropriate degree of trust.

10

Swarms in Space Missions

New NASA mission concepts now being studied involve many small spacecraft operating collaboratively, analogous to swarms in nature. The swarm concept offers several advantages over traditional large spacecraft missions: the ability to send spacecraft to explore regions of space where traditional craft simply would be impractical, greater redundancy (and, consequently, greater protection of assets), and reduced costs and risk, among others [176, 181]. Examples include

- several unmanned aerial vehicles (UAVs) flying approximately one meter above the surface of Mars, which will cover as much of the surface of Mars in minutes as the now famous Mars rovers did in their entire time on the planet
- armies of tetrahedral walkers to explore the Martian and Lunar surface and
- miniaturized pico-class spacecraft to explore the asteroid belt.

Under these concepts for future space exploration missions, swarms of spacecraft and rovers will act much like insects such as ants and bees. Swarms will operate as a large group of autonomous individuals each having simple, cooperative capabilities, and no global knowledge of the group's objective. Such systems entail a wide range of potential new capabilities but pose unprecedented challenges to system developers. Swarm-based missions, with a new level and kind of complexity that makes untenable the idea of individual control by human operators, suggest the need for a new level and kind of autonomy and autonomicity. Instead of employing human operators to individually control the members of the swarm, a completely different model of operations would be required, where the swarm will operate completely autonomously or with some control at the swarm level.

This chapter will describe swarm-based systems, the possible use of swarms in future space missions, technologies needed to implement them, and some of the challenges in developing them. We will outline the motivation for using swarms in future exploration missions. We will describe one concept mission

in relation to the characteristics that such a mission (and similar systems) would need to exhibit in order to become a reality.

10.1 Introduction to Swarms

In nature, swarms are large groupings of insects such as bees or locusts where each insect has a simple role, but where the swarm as a whole produces complex behaviors. Strictly speaking, such emergence of complex behavior is not limited to swarms, and there are similar complex social structures occurring with higher order animals and insects that don't swarm *per se*, such as colonies of ants, flocks of birds, packs of wolves, etc. The idea that swarms can be used to solve complex problems has been taken up in several areas of computer science. The term "swarm" in this book refers to a large grouping of simple components working together to achieve some goal and produce significant results [12]. The result of combining simple behaviors (the microscopic behavior) is the emergence of complex behavior (the macroscopic behavior) and the ability to achieve significant results as a "team" [16]. The term should not be taken to imply that these components fly (or are airborne); they may just as well operate on the surface of the earth, under the surface, under water, or in space (including other planets).

Intelligent swarm technology is based on swarm technology where the individual members of the swarm also exhibit independent intelligence [13] and thus act as agents. Intelligent swarms may be heterogeneous or homogeneous. Even if the swarm starts out as homogeneous, the individual members, with differing environments, may learn different things and develop different goals, and in this way therefore the swarm becomes heterogeneous. Intelligent swarms may also be made up of heterogeneous elements from the outset, reflecting different capabilities as well as a possible social structure.

Agent swarms are being used in computer modeling and have been used as a tool to study complex systems [55]. Examples of simulations that have been undertaken include swarms of birds [21, 115], problems in business and economics [94], and ecological systems [131]. In *swarm simulations*, each of the agents is given certain parameters that it tries to maximize. In terms of bird swarms, each bird tries to find another bird to fly with, and then flies off to one side and slightly higher to reduce its drag, and eventually the birds form flocks. Other types of swarm simulations have been developed that exhibit unlikely emergent behavior. These emergent behaviors are the sums of often simple individual behaviors but, when aggregated, form complex and often unexpected behaviors. Swarm behavior is also being investigated for use in such applications as telephone switching, network routing, data categorization, command and control systems, and shortest path optimizations.

Swarm intelligence techniques (note the slight difference in terminology from "intelligent swarms") are population-based stochastic methods used in

combinatorial optimization problems. In these models, the collective behavior of relatively simple individuals arises from local interactions between each individual and its environment and between each individual and other members of the swarm, which finally results in the emergence of global functional actions by the swarm. Swarm intelligence represents a metaheuristic approach to solving a wide range of problems.

Swarm robotics is the application of swarm intelligence techniques to robotic devices. These systems are programmed to act much like insect swarms where each robot senses and reacts to near-by robots as well as the environment with a given behavior. For example, each robot of an underwater swarm may be watching and following a neighbor but is also sensing its environment. When something of interest is found by one, it will communicate the information to its neighbors and swim toward it. The others will follow the new leader until they get to the object of interest and then swarm around and examine it. It may be that only a portion of the swarm breaks off, forming a subteam, while the others continue the swarm's search. When the subteam is finished examining the object, they rejoin the team, so there is a constant breaking off and rejoining by members of the swarm.

Swarms may also operate in a tight or loose group and move between extremes depending on the current state of the mission. The group may be tight not only physically, but also operationally. For example, during exploration the swarm may be scattered over a large area and communicate very little while they each perform their searches. Then, when one finds something of interest, it may broadcast information to inform the rest of the swarm. Others may respond regarding something similar that has already been examined and a group of swarm members may work computationally close to each other (even if they are physically separated) to determine whether it should be further investigated. If so, a subteam would be dispatched to the location and they would work cooperatively to obtain further information (physically close).

10.2 Swarm Technologies at NASA

The Autonomous Nano Technology Swarm (ANTS) project was a joint NASA Goddard Space Flight Center (GSFC) and NASA Langley Research Center (LARC) collaboration whose purpose was to develop revolutionary mission architectures and exploit artificial intelligence techniques and paradigms in future space exploration [32, 29]. This project researched mission concepts that could make use of swarm technologies for both spacecraft and surface-based rovers.

ANTS consists of a number of mission concepts that include:

SMART: Super Miniaturized Addressable Reconfigurable Technology uses miniaturized robots based on tetrahedrons to form swarms of configurable robots.

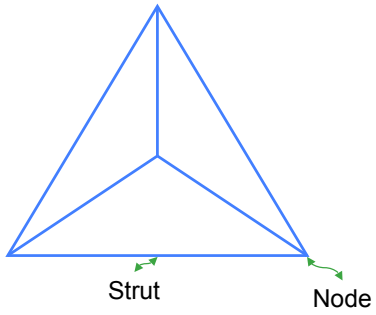

Fig. 10.1. Basic unit of tetrahedral structures.

PAM: Prospecting Asteroid Mission would also launch 1000 pico-class spacecraft with the aim of exploring the asteroid belt and collecting data on particular asteroids of interest. PAM is described below in more detail.

SARA: The Saturn Autonomous Ring Array would use 1000 pico-class spacecraft, organized as ten sub-swarms, each with specialized instruments, to perform *in situ* exploration of Saturn's rings, so as to understand their make up and how they were formed. The concept mission would require self-configuring structures for nuclear propulsion and control. Additionally, autonomous operation would be necessary for both maneuvering around Saturn's rings and collision avoidance between spacecraft.

ANTS Application Lunar Base Activities would exploit new NASA-developed technologies in the field of miniaturized robotics, which would form the basis of remote landers to be launched to the moon from remote sites, and would exploit innovative techniques (described below in Section 10.2.1) to allow rovers to move in an amoeboid-like fashion over the moon's uneven terrain.

The following sections describe the SMART and PAM mission concepts. The description of SMART covers similar technologies that would also be needed for the Lander Amorphous Rover Antenna (LARA) (and other) concept missions. Since SARA and PAM have many attributes in common (as regards autonomous operation), we will concentrate on a description of PAM in the following.

10.2.1 SMART

The ANTS SMART (Super Miniaturized Addressable Reconfigurable Technology) architectures were initiated to develop new kinds of structures capable of:

- goal-oriented robotic motion,
- changing form to optimize function (morphological capabilities),

- adapting to new environmental demands (learning and adaptation capabilities), and
- repairing-protecting itself (autonomic capabilities).

The basic unit of the structures is a tetrahedron (Figure 10.1) consisting of four addressable nodes interconnected with six struts that can be reversibly deployed or stowed. More complex structures are formed from interconnecting these reconfigurable tetrahedra, making structures that are scalable, and leading to massively parallel systems. These highly-integrated, 3-dimensional meshes of actuators/nodes and structural elements hold the promise of providing a new approach to robust and effective robotic motion. The current working hypothesis is that the full functionality of such a complex system requires fully autonomous intelligent operations at each node.

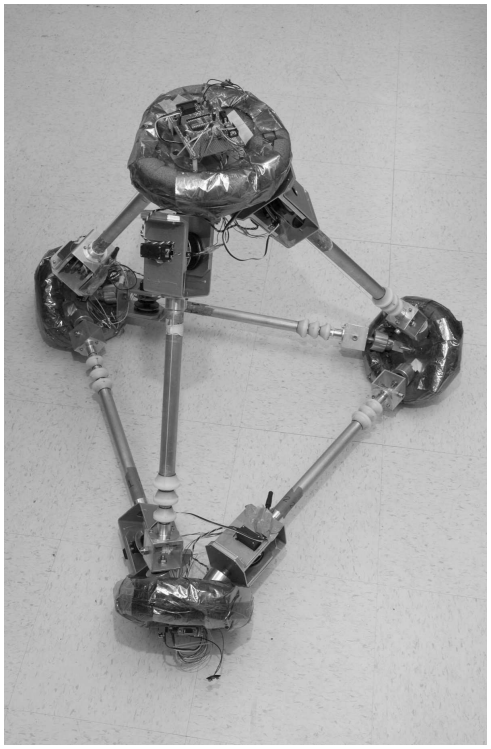

Fig. 10.2. Prototype of a Tetrahedron rover (Image Credit: NASA).

The tetrahedron (tet) "walks" by extending certain struts, changing its center of mass and "falling" in the desired direction. As the tetrahedral structure "grows" by interfacing more and more tets, the falling motion evolves to a

smoother walking capability, i.e., the smoother walking-climbing-avoiding capabilities emerge from the orchestration of the capabilities of the tetrahedra involved in the complex structure. Figure 10.2 shows a picture of a prototype tetrahedron.

The basic tetrahedron structure was modeled as a communicating and cooperating/collaborating four-agent system with an agent associated with each node of the tetrahedron. An agent, in this context, is an intelligent autonomous process capable of bi-level deliberative and reactive behaviors with an intervening neural interconnection (the structure of the neural basis function [30]). The node agents also possess social and introspective behaviors. The problem to be solved is to scale this model up to one capable of supporting autonomous operation for a 12-tet rover, a structure realized by the integration of 12 tets in a polyhedral structure. The overall objective is to achieve autonomous robotic motion of this structure. Figure 10.3 shows a drawing of a 12-tet rover.

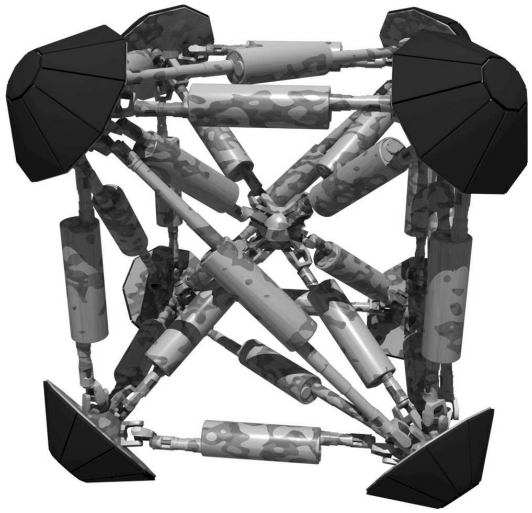

Fig. 10.3. A picture of a 12-tet rover (Image Credit: NASA).

10.2.2 NASA Prospecting Asteroid Mission

The ANTS PAM (Prospecting Asteroid Mission) concept mission [31, 32, 181] would involve the launch of a swarm of autonomous pico-class (approximately

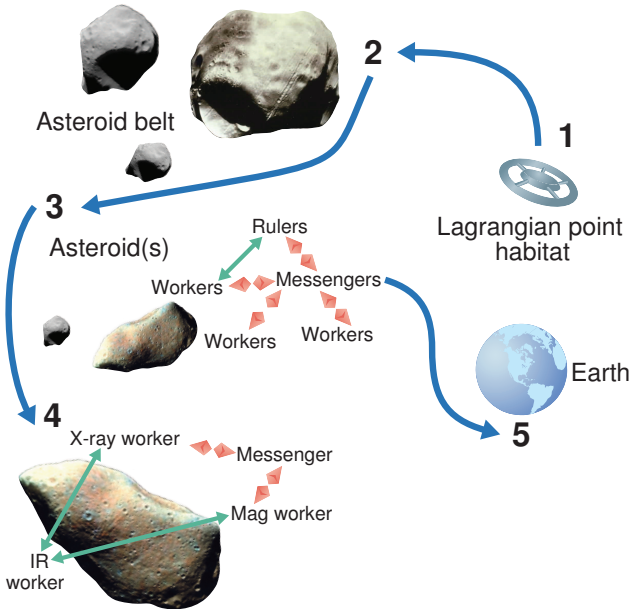

Fig. 10.4. NASA's Autonomous Nano Technology Swarm (ANTS) mission overview.

1kg) spacecraft that would explore the asteroid belt for asteroids with characteristics of scientific interest. Figure 10.4 gives an overview of the PAM mission concept [176]. In this mission, a transport ship launched from earth would travel to a Lagrangian point. From this point, 1000 spacecraft, which would have been assembled *en route* from earth, would be launched into the asteroid belt. Each spacecraft would have a solar sail as the primary means of propulsion (using photon pressure from the Sun's light), supplemented with tiny thrusters to maneuver independently. Each spacecraft would carry just one specialized instrument for collecting a specific type of data from asteroids in the belt. With onboard computational resources, each spacecraft would also have artificial intelligence and heuristics systems for control at the individual and team levels. For communications, spacecraft would use low bandwidth to communicate within the swarm and high bandwidth for sending data back to earth. It is expected that 60 to 70 percent of the spacecraft would be lost during the mission, primarily because of collisions with each other or with asteroids during exploration operations, since their ability to maneuver will be severely limited.

As Figures 10.4 and 10.5 show, teams would consist of members from three classes of spacecraft within the swarm, with members in each class combining to form teams that explore particular asteroids. Approximately 80 percent of the spacecraft would be workers that carry the specialized instruments to

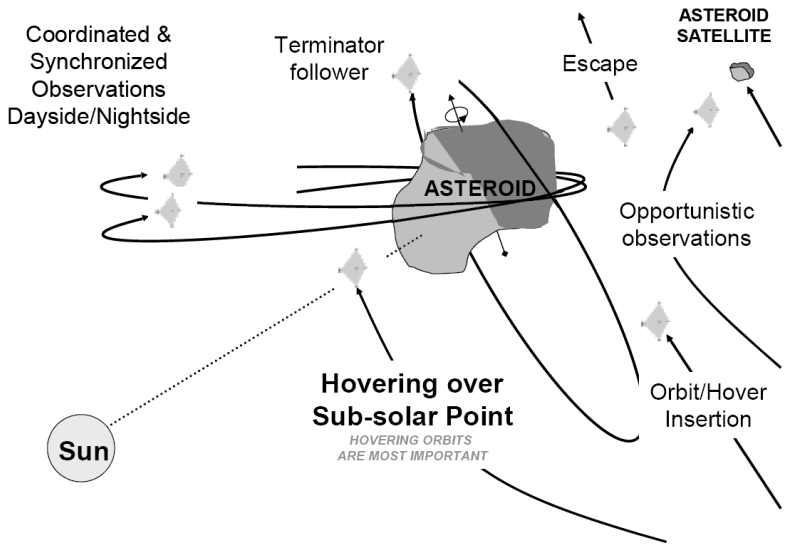

Fig. 10.5. ANTS encounter with an asteroid (Image Credit: NASA).

obtain specific types of data. Examples of instruments include magnetometers and x-ray, gamma-ray, visible/infrared, or neutral mass spectrometers. Each worker would gather only its assigned data types. Some of the spacecraft would be coordinators (called leaders or rulers) that would coordinate the efforts of the workers. They would apply rules to determine the types of asteroids and data of interest to the mission. The third type of spacecraft are called messengers. Messengers would coordinate communications among the workers, rulers, and mission control on earth.

Figure 10.5 depicts the flow of activity as teams explore an asteroid, exchange data, and return data to earth. A single ANTS spacecraft could also survey an asteroid in a flyby, sending quick-look data to the ruler, which would then decide whether the asteroid warranted further investigation using a team. The ruler would choose team members according to the instruments they carry.

Many operational scenarios are possible within the overall concept of missions that act like a natural swarm culture. In one scenario, the swarm would form sub-swarms under the control of a ruler, which would contain models of the types of science that it wants to perform. The ruler would coordinate workers, each of which would use its individual instrument to collect data on specific asteroids and feed this information back to the ruler, which would determine which asteroids are worth examining further. If after consulting its selection criteria and using heuristic reasoning it determines that the asteroid merits further investigation, an imaging spacecraft would be sent to the asteroid to ascertain its shape and size and to create a rough model to

be used by other spacecraft for maneuvering around the asteroid. The ruler would also arrange for additional workers to transit to the asteroid with an expanded repertoire of instruments to gather more complete information. In effect, the spacecraft would form a team. The leader would be the spacecraft that contains models of the types of experiments or measurements the team needs to perform. The leader would relay parts of this model to the team workers, which then would take measurements of asteroids using whatever type of instrument they have until something matched the goal the leader sent. The workers would gather the required information and send it to the team leader, which would integrate it and return it to the ruler that formed the team. The ruler might then integrate this new information with information from previous asteroid explorations and use a messenger to relay findings back to earth.

10.2.3 Other Space Swarm-Based Concepts

An autonomous space exploration system studied at Virginia Tech, funded by the NASA Institute for Advanced Concepts (NIAC), consists of a swarm of low altitude, buoyancy-driven gliders for terrain exploration and sampling, a buoyant oscillating wing that absorbs wind energy, and a docking station that could be used to anchor the energy absorber, charge the gliders, and serve as a communications relay [97]. The work built on success with underwater gliders used for oceanographic research. The intent was to develop low-cost planetary exploration systems that could run autonomously for years in harsh environments, such as in the sulfuric acid atmosphere of Venus or on Titan (the largest of Saturn's moons).

A second NASA swarm-related project titled "Extremely Large Swarm Array of Picosats for Microwave/RF Earth Sensing, Radiometry, and Mapping" [73] was also funded by NIAC. The proposed telescope would be used to do such things as characterize soil moisture content, atmospheric water content, snow accumulation levels, flooding, emergency management after hurricanes, weather and climate prediction, geological feature identification, and others. To accomplish this would require an antenna size on the order of 100 kilometers at a GEO orbit. To implement such an large antenna, a highly sparse spacefed array antenna architecture was proposed that would consist of 300,000 picosats, each being a selfcontained onechip spacecraft weighing 20 grams.

10.3 Other Applications of Swarms

The behavior of swarms of bees has also been studied as part of the BioTracking project at Georgia Tech [9]. To expedite the understanding of the behavior of bees, the project videotaped the behavior of bees over a period of time, using a computer vision system to analyze data on sequential movements that

bees use to encode the location of supplies of food, etc. It is anticipated that such models of bee behavior can be used to improve the organization of cooperating teams of simple robots capable of complex operations. A key point is that the robots need not have *a priori* knowledge of the environment, and that direct communication between robots in the teams is not necessary.

Eberhart and Kennedy have developed an optimization technique based on particle swarms [79] that produces fast optimizations for a wide number of areas including UAV route planning, movement of containers on container ships, and detecting drowsiness of drivers. Research at Penn State University has focused on the use of particle swarms for the development of quantitative structure activity relationships (QSAR) models used in the area of drug design [23]. The research created models using artificial neural networks and k-nearest neighbor and kernel regression. Binary and niching particle swarms were used to solve feature-selection and feature-weighting problems.

Particle swarms have influenced the field of computer animation also. Rather than scripting the path of each individual bird in a flock, the Boids project [115] elaborates a particle swarm of simulated birds. The aggregate motion of the simulated flock is much like that in nature. The result is from the dense interaction of the relatively simple behaviors of each of the (simulated) birds, where each bird chooses its own path.

Much success has been reported from the use of Ant Colony Optimization (ACO), a technique that studies the social behaviors of colonies of ants, and uses these behavior patterns as models for solving difficult combinational optimization problems [35]. The study of ants and their ability to find shortest paths has lead to ACO solutions to the traveling salesman problem, as well as network and internet optimizations [34, 35].

Work at University of California Berkeley is focusing on the use of networks of Unmanned Underwater Vehicles (UUVs). Each UUV has the same template information, containing plans, subplans, etc., and relies upon this and its own local situation map to make independent decisions, which will result in cooperation between all of the UUVs in the network. Experiments involving strategies for group pursuit were also done.

10.4 Autonomicity in Swarm Missions

Swarms are being used in devising solutions to various problems principally because they present an appropriate model for those problems. Sections 10.2 and Section 10.3 described several application areas of swarm technology where the approach seems to be particularly successful.

But swarms (in nature or otherwise) inherently need to exhibit autonomic properties. To begin with, swarms should be self-directed and self-governing. Recall that this is achieved through the complex behavior that emerges from the combination of several simple behaviors and their interaction with the environment. It can be said that in nature, organisms and groups/colonies

of individuals, with the one fundamental goal of survival, would succumb as individuals and even as species without autonomicity. A natural conclusion is that artificial swarms with planned mission objectives must also possess autonomicity.

The described ANTS PAM concept mission would need to exhibit almost total autonomy to succeed. The mission would also exhibit many of the properties required to qualify it as an autonomic system [161, 181, 182]:

Self-Configuring: The ANTS' resources must be fully configurable to support concurrent exploration and examination of hundreds of asteroids. Resources must be configured at both the swarm and team (sub-swarm) levels in order to coordinate science operations while simultaneously maximizing resource utilization.

Self-Optimizing: Rulers self-optimize primarily through learning and improving their ability to identify asteroids that will be of interest to scientists. Messengers self-optimize through positioning themselves appropriately for optimum communications. Workers self-optimize through learning and experience. Self-optimization at the swarm level propagates up from the self-optimization of individuals.

Self-Healing: ANTS must self-heal to recover from damage due to solar storms or collisions with an asteroid or between ANTS spacecraft. Loss of a ruler or messenger may involve a worker being "upgraded" to fulfill that role. Additionally, loss of power may require a worker to be listed as dead ("killed off" via an apoptosis mechanism [161, 145]).

Self-Protecting: In addition to protecting themselves from collision with asteroids and other spacecraft, ANTS teams must protect themselves from solar storms, where charged particles can degrade sensors and electronic components and destroy solar sails (the ANTS spacecraft's sole source of power and primary means to perform maneuvering). ANTS teams must re-plan their trajectories or, in worst-case scenarios, must go into "sleep" mode to protect their sails and instruments and other subsystems.

The concept of autonomicity can be further elaborated beyond the self-chop properties listed above. Three additional self-properties—self-awareness, self-monitoring, and self-adjusting—will facilitate the basic self-properties. Swarm (ANTS) individuals must be aware (have knowledge) of their own capabilities and limitations, and the workers, messengers, and rulers will all need to be involved in constant self-monitoring and (if necessary) self-adjusting, thus forming a feedback control loop. Finally, the concept of autonomicity would require environmental awareness. The swarm (ANTS) individuals will need to be constantly environmentally aware to enable effective self-adaptation and ensure mission success.

10.5 Software Development of Swarms

Developing the software for the ANTS missions would be monumentally complicated. The total autonomy requirement would mean that the software would likely be based on a heuristic approach that accommodates the swarm's social structure. Artificial-intelligence technologies, such as genetic algorithms, neural nets, fuzzy logic, and on-board planners are candidate solutions. But the autonomic properties, which alone make the system extremely complex, are only part of the challenge. Add intelligence for each of the thousand interacting spacecraft, and it becomes clear that the mission depends on several breakthroughs in software development.

10.5.1 Programming techniques and tools

A primary requirement would be a new level or new class of programming techniques and tools that either replace or build on object-oriented development. The idea is to reduce complexity through novel abstraction paradigms that would essentially "abstract away" complexity. Developers would use predefined libraries or components that have been solidly tested and verified. The level of programming languages would need to be high enough that developers could use constructs that are natural extensions to the software type under development.

Another requirement would be tools and techniques that would have built-in autonomic, intelligent, and interacting constructs to reduce development time and increase developer productivity. Tools would need to allow rapid simulation so that developers might identify errors in requirements or code at the earliest stage possible. For now, ideas about creating standard intelligent, autonomic components are still evolving: there is yet no consensus as to what constitutes a system of such components. Hopefully, more research and development in these areas will yield effective and timely results.

10.5.2 Verification

These new approaches to exploration missions simultaneously pose many challenges. Swarm missions will be highly autonomous and will have autonomic properties. Many of these missions will be sent to parts of the solar system where manned missions are regarded as infeasible, and where in some instances the round-trip delay for communications between earth and the spacecraft exceeds 40 minutes, meaning that the decisions on responses to problems and undesirable situations must be made *in situ* rather than from ground control on earth. The degree of autonomy that such missions will require would mean an extreme burden of testing in order to accomplish system verification. Furthermore, learning and adaptation towards continual improvements in performance during mission operations will mean that emergent behavior patterns simply cannot be fully predicted through the use of traditional

system development methods. Consequently, formal specification techniques and formal verification will play vital roles in the future development of these types of missions.

Full testing of software of the complexity of the ANTS mission may be recognized as a heavy burden and may have questionable feasibility, but verification of the on-board software, especially the mechanism that endows the spacecraft with autonomy and the ability to learn, is crucial because the one-way communications delay makes real-time control by human operators on earth infeasible. Large communications delays mean human operators could not, in many scenarios, learn of problems or errors or anomalies in the mission until the mission had substantially degraded or failed. For example, in a complex system with many concurrently communicating processes on board or among the members of the swarm, race conditions are highly likely—but such conditions rarely come to light during the testing or mission development phase by inputting sample data and checking results. These types of errors are time based, occurring only when processes send or receive data at particular times or in a particular sequence, or after learning takes place. To find these errors, testers must execute the software in all the possible combinations of the states of the communicating processes. Because the state space is extremely large (and probably extremely difficult to project in sufficient detail for actual testing), these systems become untestable with a relatively small number of elements in the swarm. Traditionally, to get around the state-space explosion problem, testers have artificially reduced the number of states of these types of systems and approximated the underlying software using models. This approach in general sacrifices fidelity and can result in missed errors. Consequently, even with relatively few spacecraft, the state space can be too large to realistically test.

One of the most challenging aspects of using swarm technology is determining how to verify that emergent system behavior will be proper and that no undesirable behaviors will occur. Verifying intelligent swarms is even more difficult, because the swarms no longer consist of homogeneous members with limited intelligence and communications. Verification will be difficult not only because each individual is tremendously complex, but also because of the many interacting intelligent elements. To address the verification challenge, ongoing research is investigating formal methods and techniques for verification and validation of swarm-based missions.

Formal methods are proven approaches for ensuring the correct operation of complex interacting systems. Formal methods are particularly useful in specifying complex parallel and distributed systems—where a single person finds it difficult to fully understand the entire system and where there are typically multiple developers. Testers can use a formal specification to prove that system properties are correct—for example, that the underlying system will go from one state to another or not into a specific state. They can also check for particular types of errors, such as race conditions, and use the formal specification as a basis for model checking.

Most formal methods do not address the problem of verifying emergent behavior. Clearly in the ANTS PAM concept, the combined behavior of individual spacecraft is far more complex than the behavior of each spacecraft in isolation. The FAST (Formal Approaches to Swarm Technologies) project surveyed formal methods techniques to determine whether any would be suitable for verifying swarm-based systems and their emergent behavior [125, 126]. The project found that there are a number of formal methods that support the specification of either concurrency or algorithms, but not both. Though there are a few formal methods that have been used to specify swarm-based systems, the project found only two formal approaches that were used to analyze the emergent behavior of swarms.

Weighted Synchronous Calculus of Communicating Systems (WSCCS), a process algebra, was used by Sumpter, et al., to analyze the non-linear aspects of social insects [163]. X-Machines have been used to model cell biology [61, 62] and, with modifications, the X-Machines model has the potential for specifying swarms. Simulation approaches are also being investigated to determine emergent behavior [55]. However, these approaches do not predict emergent behavior from the model, but rather model the emergent behavior after the fact.

Formal Approaches to Swarm Technology The FAST project defined an integrated formal method, which is appropriate for the development of swarm-based systems [121]. Future work will concentrate on the application of the method to demonstrate its usefulness, and on the development of appropriate support tools. NASA is pursuing further development of formal methods techniques and tools that can be applied in the development of swarm-based systems to help achieve confidence in their correctness.

10.6 Future Swarm Concepts

A brief overview of swarm technologies was presented with emphasis on their relevance for potential future NASA missions. Swarm technologies hold promise for complex exploration and scientific observational missions that require capabilities that would be unavailable in missions designed around single spacecraft.

While swarm autonomy is clearly essential for missions where human control is not feasible (e.g., when communications delays are too great or communications data rates are inadequate for effective remote control), autonomicity is essential for survival of individual spacecraft as well as the entire swarm as a consequence of hostile space environments.

Although ANTS was a concept mission, the underlying techniques and technologies that were developed are also motivating other technologies and applications. ANTS technology has many potential applications in military and commercial environments, as well as in other space missions. In military surveillance, smaller craft, perhaps carrying only a basic camera or other

instrument, could coordinate to provide 3D views of a target. The US Navy has been studying the use of vehicle swarms for several years. In mining and underwater exploration, autonomous craft could go into areas that are too dangerous or small for humans. For navigation, ANTS technology could make GPS cheaper and more accurate because using many smaller satellites for triangulation would make positioning more accurate.

Finally, in other types of space exploration, a swarm flying over a planetary surface could yield significant information in a short time. The ANTS technology could also benefit commercial satellite operations, making them both cheaper and more reliable. With its autonomic properties, a swarm could easily replace an individual pico-satellite, preserving operations that are now often lost when satellites become damaged. Mission control could also increase functionality simply by having the swarm add members (perhaps from a collection of pico-satellites already in orbit as standby spares) with the needed functionality, rather than launching a new, large, complex satellite.

The obvious need for advances in miniaturization and nanotechnology is prompting groundbreaking advances at NASA and elsewhere. The need for more efficient on-board power generation and storage motivates research in solar energy and battery technology, and the need for energy-efficient propulsion motivates research on solar sails and other technologies such as electric-field propulsion. The ANTS concepts also push the envelope in terms of software technologies for requirements engineering, nontrivial learning, planning, agent technology, self-modifying systems, and verification and validation technologies. The paradigms, techniques, and approaches stimulated by concept missions like ANTS open the way for new types of future space exploration missions: namely, large numbers of small, cooperating spacecraft conducting flexible, reliable, autonomous, and cost-effective science and exploration operations beyond the capabilities of the more familiar large spacecraft.

11

Concluding Remarks

In this book, we have examined technologies for system autonomy and autonomicity. We have considered what it means for a system to be autonomous and autonomic, and have projected how the concepts might be applied to spacecraft and other aspects of NASA missions. We discussed current spacecraft ground and flight operations, described how autonomy and autonomic technology is currently applied to NASA missions, and identified the areas where additional autonomy could be beneficially effective. We also considered artificial intelligence techniques that could be applied to current and future missions to provide additional autonomy and autonomicity.

We now proceed to identify factors that drive the use of new technology and discuss the necessity of software reliability for space missions. We will discuss certain future missions and their needs for autonomy and autonomic technology, and finally will consider the NASA strategic plan and the manner in which autonomy and autonomic systems may be involved in supporting that plan.

11.1 Factors Driving the Use of Autonomy and Autonomicity

As discussed in other parts of this book, there are a number of factors that drive the application of autonomy and autonomicity. New science using space-based platforms gives rise progressively to new methods and means, and calls for new and increasingly sophisticated instruments and complex new instrument configurations in space, as will be seen from a number of examples to be given later. Future explorations in more remote environments bring new challenges for mission control when real-time information for human operators becomes impossible due to the fundamental reality of signal delays under speed-of-light limitations. Reductions of mission costs is a continuing and evidently unavoidable necessity, and as we have seen has been increasingly realized through reducing human involvement in routine mission operations—a

likely recurring theme in future missions. These are among the principal factors that make autonomy and autonomicity an increasing necessity in many future space missions.

New scientific discovery is enabled by observing deeper into space using instruments that are more sensitive and complex, by making multiple observations simultaneously with coordinating and cooperating spacecraft, and by reacting to the environment or science of opportunity more quickly. All of these capabilities can be realized either through the use of autonomous systems and autonomic properties or by having a human onboard the spacecraft or, except in the case of very remote assets, by having a sufficiently large operations staff at the mission control center. In some missions, a human onboard the spacecraft would not be able to react fast enough to a phenomenon, or maintain required separations between spacecraft. In other missions, such as missions to another planet or the asteroid belt, a crewed spacecraft would be infeasible, too dangerous, or too costly.

We also saw in Chapter 3 that adding autonomy to missions is not new. Autonomy or automation has been an increasing aspect of flight and ground software with the gradually increasing complexity of missions and instruments and with ongoing budgetary pressures. This trend is continuing into future NASA missions, more particularly robotic (un-crewed) missions.

11.2 Reliability of Autonomous and Autonomic Systems

In NASA missions, software reliability is extremely important. A software failure can mean the loss of an entire mission. Ground based systems can be tended by humans to correct any errors. In unmanned space systems, with only a few exceptions (e.g., the Hubble Space Telescope, which was designed to be serviced on orbit by space-walking humans), any corrections must be performed strictly via radio signals, with no possibility of human presence. Therefore, software and hardware for space missions must be developed, verified, and tested to a high level of assurance, with a corresponding cost in time and money.

Because of the need for high assurance, one of the challenges in adding autonomy and autonomicity to spacecraft systems is to implement these concepts so they work reliably and are verifiable. The software must be robust enough to run on a spacecraft and perhaps as part of a community of spacecraft. Further, the software must be implementable in a reasonable timeframe and for a reasonable cost (relative to the type and importance of the mission).

Autonomous systems often require flexible communication systems, mobile code, and complex functionality, not all of which is always fully understood at the outset. A particular problem with these types of systems is that such systems can never really be tested to any degree of sufficiency, as an intelligent system may adapt its behavior on every execution. New ways of testing and

monitoring this type of software are needed to give mission principal investigators the assurance that the software is correct [117]. This was addressed to some extent in a previous book in this series [119].

In addition to being space-based, many of the proposed missions will operate very remotely and without frequent contact with a ground-based operations control center or with a large communications lag time. Such conditions make detecting and correcting software errors before launch even more important, because patching the software during the mission's operational phase will be difficult, impractical, or impossible.

Autonomous missions are still at a relatively early stage of evolution in NASA, and the software development community is still learning approaches to their development. These are highly parallel systems that can have very complex interactions. Even simple interacting systems can be difficult to develop, as well as debug, test, and validate. In addition to being autonomous and highly parallel, these missions may also have intelligence built into them, and they can be distributed and can engage in asynchronous communications. Consequently, these systems are difficult to verify and validate.

New verification and validation techniques are required [101, 113, 118, 120, 124]. Current techniques, based on large monolithic systems, have worked well and reliably, but do not translate to these new autonomous systems, which are highly parallel and nondeterministic.

11.3 Future missions

Some future missions were discussed in other parts of this book. The following material from indicated sources describes additional mission concepts that are undeniably ambitious in nature—the success of which will require autonomous and autonomic properties (Table 11.1).[1]

Mission	Launch Date	Number of Spacecraft
Big Bang Observer	2020+	Approx. 12
Black Hole Imager	2025+	Approx. 33
Constellation X	2020+	4
Stellar Imager	after 2020	Approx. 17
LISA	2018	3
MIDEX SIRA	2015+	12–16
Enceladus	2018	2
Titan	2018	2
JWST	2013+	1

Table 11.1. Some NASA Future missions.

[1]The following descriptions are summarized from the NASA future missions web site

The Laser Interferometer Space Antenna (LISA) mission concept specifies spacecraft that measure passing gravitational waves. LISA will be a constellation of three spacecraft that uses laser interferometry for precise measurement of distance changes between widely separated freely falling test masses housed in each spacecraft (Figure 3.3). The spacecraft are at the corners of an approximately equilateral triangle about five million kilometers on a side in heliocentric orbit. The science instrument is created via laser links connecting the three spacecraft. It is formed by measuring to high levels of precision the distances separating the three spacecraft (i.e., the test masses) via the exchange of the laser light. From the standpoint of Bus FSW providing spacecraft attitude and position control, the number of sensors and actuators that must be interrogated and commanded is at least twice the number associated with a more traditional mission. Similarly, the number of control modes is double that of a typical astrophysics mission, as are the number of parameters solved-for by the state estimator.

The Big Bang Observer and the Black Hole Imager are part of the Beyond Einstein program that will further test Einstein's general theory of relativity. The Big Bang Observer will explore the beginning of time and will build on the LISA mission to directly measure gravitons from the early Universe still present today. The Black Hole Imager mission will calculate the aspects of matter that fall into a black hole and do this by conducting a census of hidden black holes, revealing where, when, and how they form.

The Constellation-X mission, a combination of several X-ray satellites orbiting in a constellation, will explore the behavior of space and time in extreme gravity. It will measure the spectral signatures of gas swirling into black holes, and the LISA will record the tune to which black holes dance around each other. There is no substitute, however, for a direct image. A Black Hole Imager, based on a technique known as X-ray interferometry, would be able to take this epochal picture, revealing a dark spot representing the outer boundary (event horizon) of a black hole.

The Stellar Imager mission will help increase understanding of solar/stellar magnetic activity and its impact on the origin and continued existence of life in the Universe, structure and evolution of stars, and habitability of planets. It will also study magnetic processes and their roles in the origin and evolution of structure and the transport of matter throughout the Universe. The current baseline architecture for the full Stellar Imager mission is a space-based, UV-Optical Fizeau Interferometer with 20–30 one-meter primary mirrors, mounted on formation-flying "mirrorsats" distributed over a parabolic virtual surface whose diameter can be varied from 100 m up to as much as 1000 m, depending on the angular size of the target to be observed (Figure 11.1). The hub and all of the mirrorsats are free-flyers in a tightly-controlled formation in a Lissajous orbit around the Sun-Earth Lagrange L2 point. The mission will also use autonomous analysis of wavefronts and will require real-time correction and control of tight formation flying.

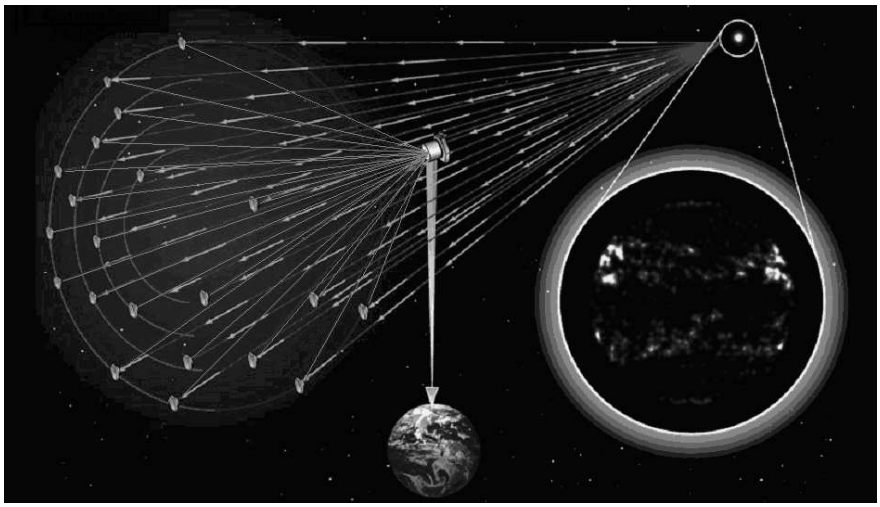

Fig. 11.1. The Stellar Imager mission (Image Credit: NASA).

The Solar Imaging Radio Array (SIRA) mission will be a Medium Explorer (MIDEX) mission and will perform interferometric observations of low frequency solar and magnetospheric radio bursts. The primary science targets are coronal mass ejections (CMEs), which drive radio-emission-producing shock waves. A space-based interferometer is required, because the frequencies of observation (<15 MHz) do not penetrate the ionosphere. SIRA will require 12 to 16 microsatellites to establish a sufficient number of baselines with separations on the order of kilometers. The microsat constellation consists of microsats located quasi-randomly on a spherical shell, initially of radius 5 km. The baseline microsat is 3-axis stabilized with an earth-pointing body-mounted high gain antenna and an articulated solar array. The micro-sats will have limited inter-mirosat communications to help maintain prescribed distances each to the others.

A mission to Enceladus is one of several options NASA is considering for an outer planets Flagship mission that would launch no earlier than 2015. Several types of missions are being studied for Enceladus. One is where a lander is sent down to collect samples. The lander would use autonomous hazard avoidance to land safely. The autonomous hazard avoidance would use a descent camera to identify and avoid rocks and blocks of ice in order to land at a relatively smooth and flat location. The lander would perform its observations, collect the samples, and analyze them, while the orbiter continues to orbit Saturn, beyond communication range. When the orbiter returns 8.22 days later, the lander would uplink its data to the orbiter and conclude its mission.

The Titan Explorer with Orbiter mission will map Titan with a high-resolution radar and study the atmosphere, pre-biological chemistry, and potential life. The mission will include an orbiter to relay communications from a module that will land on the surface. The orbiter will also perform aerocapture to test the atmoshpere.

The James Webb Space Telescope (JWST) mission is the replacement for the Hubble Space Telescope (Figure 11.2). The JWST flight and ground system will be developed as an integrated system that will provide seamless operations from science proposal to data distribution, with a minimum of interfaces. The spacecraft will incorporate greater on-board autonomy, executing a high level list of observations rather than a detailed timeline. This will simplify the ground system scheduling activities and permit the spacecraft to perform over many days with little or no direction from the ground. JWST will be able to operate autonomously for periods of several days between these up-links, and therefore continuous staffing of the control center will not be required in the latter years of operations. The observatory will operate nearly autonomously throughout its science operations phase. Control of most housekeeping and science collecting functions will be provided on-board the observatory by an event-driven command system. During the science operations phase of the mission, one communications contact per day and one command load per week or two will be sufficient to support operations. Figure 11.3 shows an early design of autonomous attitude control transitions for JWST [74].

11.4 Autonomous and Autonomic Systems in Future NASA Missions

The NASA 2006 Strategic Plan states:

> NASA also will develop and test technologies for power and autonomous systems that can enable more affordable and sustainable space exploration by reducing both consumables launched from earth and the risks for mission operations. Advanced power systems, including solar, fuel cell, and potential nuclear power, will provide abundant power to a lunar outpost so that exploration will not be limited by the available energy. Intelligent robotics will assist the crew in exploring, setting up, operating, and maintaining the outpost. Autonomous systems will reduce mission risk by alerting the crew to impending failures, automatically reconfiguring in response to changing conditions, and performing hazardous and complex operations.

The plan continues:

> Therefore, NASA's long-term Earth science plan is to use sentinel orbits (e.g., Lagrange points, geostationary, and medium Earth orbit)

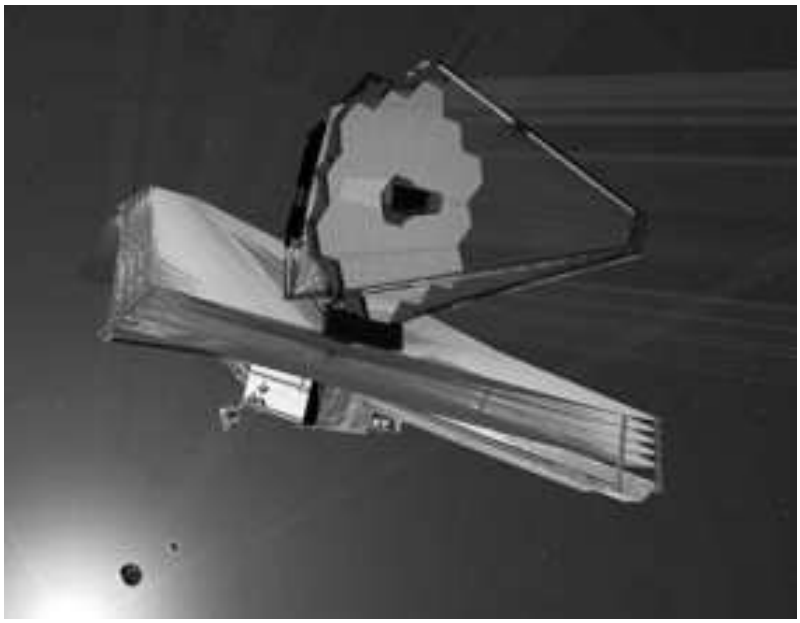

Fig. 11.2. The James Web Space Telescope (Image Credit: NASA).

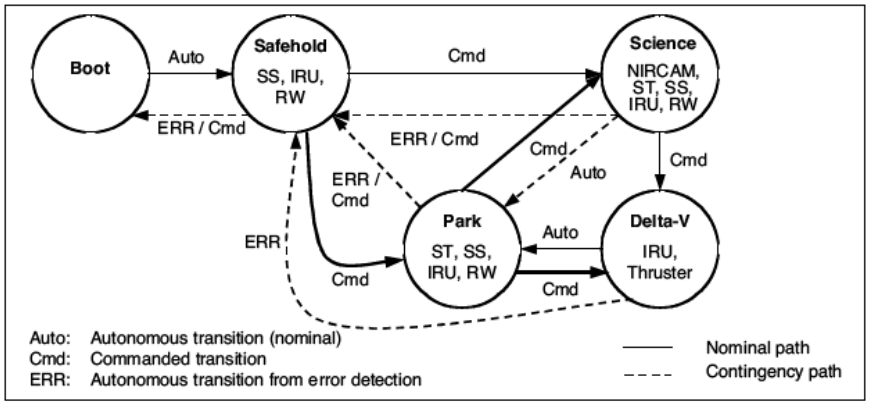

Fig. 11.3. Early design of autonomous attitude control mode transitions.

> and constellations of smart satellites as parts of an integrated, interactive "sensorweb" observing system that complements satellites in low Earth orbit, airborne sensors, and surface-based sensors. NASA will mature active remote sensing technologies (radars and lasers) to take global measurements of Earth system processes from low and geostationary Earth orbits.
>
> As new types of Earth observations become available, information systems, modeling, and partnerships to enable full use of the data for scientific research and timely decision support will become increasingly important. The sensorweb observing systems of the future will perform satellite constellation management, automated detection of environmental phenomena, tasking of other elements of the observing network, on-board data processing, data transmittal, and data archival and distribution to users of earth observations. The sensorweb will be linked to "modelwebs" of prediction systems enabled by NASA and formed by Agency partners to improve the forecast services they provide. NASA's investment in these areas (through such means as the Advanced Information Systems Technology program) will help the Nation take full advantage of enhanced information availability. In particular, the role of models in converting the satellite-produced information into useful products for environmental characterization and prediction will become more crucial.

The above statements in the NASA strategic plan project the increasing importance of mission autonomy and autonomicity. In the first quote, the strategic plan notes that autonomous systems are needed to make future missions possible and make them affordable. It further indicates the need for adding intelligence to robotics and reducing mission risks through autonomy, by, for example, alerting the crew to failures, and automatically reconfiguring (autonomicity) as mission conditions change.

The second quoted passage describes the planned sensor web in terms of constellations of smart satellites, with the prospect of automatic science of opportunity, and with heterogenous systems working together to make up the sensor web. Again, all of this will require autonomous and autonomic systems.

With the new mission concepts that are taking shape, an excellent opportunity now exists to insert autonomy and autonomicity, as well as agent technologies, into these missions. Since these technologies make many of the missions feasible, the science community is now looking to the artificial intelligence and agent software community to implement these ideas in future flight software.

This book has attempted to provide background on NASA ground and space systems and exploration thrusts, and has presented autonomy and autonomicity as a technological means to enhance space mission usefulness, cost effectiveness, and functionality. The motivation behind this book has been to

help others direct their research and development into this area, as well as to stimulate future missions to adopt more autonomy and autonomicity and thereby enhance new exploration and make scientific discovery more productive.

A

Attitude and Orbit Determination and Control

In order to perform its science mission, a spacecraft must, in general, know its orientation in space and its position relative to the targets it plans to observe. The term *orbit* refers to the spacecraft's position and velocity with respect to an inertial reference frame, when the spacecraft is not experiencing a net force from its propulsion system. The term *attitude* refers to the spacecraft's orientation with respect to some inertial reference frame (for Earth-orbiting spacecraft, this ordinarily is the geocentric inertial (GCI) frame).

Traditionally, the determination of a spacecraft's orbit was exclusively a ground system function. While in communication with the spacecraft, the ground system would collect tracking data, and then the Flight Dynamics computational facility (in the context of NASA/Goddard missions) processed the data to calculate the definitive (i.e., actual) spacecraft orbit associated with the time(s) corresponding to the collected tracking data. Flight Dynamics then would calculate a *predicted* spacecraft orbit by combining the definitive orbit data with mathematical models describing the gravitational interactions and orbital perturbations experienced by the spacecraft. If the spacecraft's flight software (FSW) did not include these models, the ground system would uplink a set of "fit parameters" (or a position/velocity-vector file) tailored to the FSW's orbit propagator (or orbit interpolator), which the spacecraft would then use to compute its position and velocity. If the spacecraft FSW did include at least a subset (or simplified version) of these models, the ground system would uplink "seed" vectors that the FSW would utilize as starting input when integrating the spacecraft equations of motion to calculate its position and velocity at some arbitrary time. In either case, the FSW output was a predicted orbital position and velocity as opposed to a measured orbital position and velocity. More recently, for low Earth orbit (LEO) spacecraft, orbital position can be directly measured to high accuracy using the onboard GPS system, which also can be used to synchronize the spacecraft clock to GPS time.

By contrast, measuring spacecraft attitude has been a standard onboard function practically since flight computers were introduced. What has evolved

over time has been the accuracy and sophistication with which those measurements have been made. In early spacecraft, onboard storage and CPU capabilities limited onboard attitude calculations to coarse attitude determination using sensors such as Sun sensors and magnetometers, or limited fine-attitude determination to processing star-tracker output using very small onboard star catalogs, or using reference stars carefully pre-selected by the ground system. In the 1990s, more powerful onboard processing capabilities and cheap onboard storage facilitated the enhancement of this capability. For example, for the RXTE mission, the FSW included a star catalog and star identification capabilities permitting the FSW to calculate the spacecraft's own fine attitude without ground support. Later still, the WMAP mission's star tracker contained its own star catalog and star identification algorithms, and so could output the spacecraft attitude directly (although it did not compensate for the star tracker's misalignment relative to the spacecraft), providing a "Lost in Space" capability.

Knowing the spacecraft position and orientation is only half of the Guidance, Navigation, and Control (GN&C) problem. The other half is controlling the spacecraft position and attitude such that the desired science can be performed. As with orbit determination, planning orbit maneuvers historically has been solely a ground function because of the CPU intensive and look-ahead nature of the problem. According to the traditional paradigm (again in the NASA/Goddard context), after determining the current spacecraft orbit, Flight Dynamics would evaluate whether that orbit satisfied the mission requirements, and, if it did, would calculate when orbit perturbations were likely to drive the orbit outside those requirements. In collaboration with the flight operations team (FOT), Flight Dynamics would then plan and schedule (as necessary) a small stationkeeping orbit maneuver designed to restore the orbit to its operational geometry. The stationkeeping plan created by this process would include highly detailed instructions (i.e., which thrusters to fire, how long they should fire, the thruster configuration when firing, etc.), which would be uplinked to the spacecraft as "Delta-V" (change-of-velocity) commands, and would be executed open-loop by the FSW, which would have no way to evaluate the success/failure of the orbit maneuver. Lastly, Flight Dynamics would evaluate the post-burn orbit to determine whether further correction were necessary. This same basic procedure also applied to the major orbit maneuvers required to acquire mission orbit, except the planning and scheduling would be more elaborate, typically requiring several burns to complete. Recently, however, serious consideration has been given to planning and scheduling routine stationkeeping orbit maneuvers autonomously onboard, as for example for the Global Precipitation Measurement (GPM) mission. Migrating this function to today's more capable flight computers not only would reduce the cost associated with using ground staff to perform a routine operation, but also would reduce operational risk by eliminating unnecessary command uplinks.

Onboard control of attitude, by contrast, has been a necessary part of all space missions other than a handful of generally low-cost missions utilizing passive control methods such as gravity gradient stabilization. Using attitude sensor measurements to determine the current attitude, the FSW compares that attitude to the commanded (i.e., desired) attitude and determines an attitude error. For spacecraft employing gyros (an attitude sensor that measures the change in spacecraft attitude during a set time period, as opposed to measuring the spacecraft's absolute orientation with respect to inertial space), a Kalman filter usually is utilized both to calculate the current attitude and to calibrate the gyro's drift bias (which ramps with time) relative to an absolute attitude sensor, such as a star tracker. The attitude error is estimated and fed into a control law that calculates on each control cycle what attitude actuator commands (e.g., reaction-wheel control torques) must be generated in order to null the error. On the next control cycle, feedback from the attitude sensors provides the information needed to determine how good a job of reducing attitude error the previous cycle's actuator commands did, as well as how much new attitude error has been introduced this cycle by external perturbative torques. Although this description of onboard attitude control has implicitly addressed maintenance of a constant commanded attitude in the presence of perturbative torques, it can equally well be applied to the execution of large desired attitude changes, called *slews*. Slews can be dealt with two ways. First, the FSW can calculate the amount of attitude change to be performed during a given control cycle, and modify the previous commanded attitude to reflect that change, which would then be used directly as part of that control cycle's attitude error. A second approach is simply to make the commanded attitude the slew target attitude. Although the control law would not be able to null that very large error (say, 90 degrees) in one control cycle, by limiting the size of the commanded control torques in a given control cycle the FSW could gradually work off the error over a series of control cycles, eventually reaching the slew's target attitude.

B

Operational Scenarios and Agent Interactions

To show more fully how the Remote Agent implementation introduced in Chapter 6 would work in an actual on-orbit situation in the uncrewed science-mission context, this chapter provides a series of operational scenarios that illustrate the interaction of Agents among themselves onboard, the interaction of flight-based Agents with ground-based Agents, and the interaction of members of a spacecraft constellation with each other.

B.1 Onboard Remote Agent Interaction Scenario

To illustrate the behavior of flight system Remote Agents (incorporating the full FSW subsystems and functionality discussed previously) cooperating to achieve a mission objective, consider the operational scenario defined by the following somewhat simplified assumptions:

1. The mission type is a Lagrangian-L2 celestial pointer.
2. The primary mission goal is to observe all ground-specified targets while minimizing fuel expenditure so as to maximize mission lifetime.
   a) The ground will group observations in clusters
   b) The ground defines the nominal order in which the observations within a cluster are performed
   c) The FSW defines the cluster order, subject to the following restrictions:
      i. If not prohibited by another restriction, on completion of all observations within a cluster, transition to the cluster whose first observation is closest to the final spacecraft attitude on completion of observations within the current cluster.
      ii. If the new cluster's observations cannot be completed before an angular-momentum dump is required, select the nearest cluster that can be completed, subject to the momentum restriction.

   iii. If no cluster satisfies restriction (2), perform an angular momentum dump and slew to the nearest remaining unobserved cluster.
   iv. If an angular-momentum dump is needed while still observing within a cluster, pause science and execute the dump on current exposure completion.

3. The secondary mission goal is to survey opportunistically ground-specified areas of the celestial sphere for new targets. These areas have been subdivided by the ground into small regions of equal size and shape.
   a) The FSW may schedule surveys of a region if no part of the region is further than a database-specified angle from the current pointing direction.
   b) No more than 2 hours out of any 24-hour period may be spent surveying.
   c) If an angular-momentum dump must be performed beforehand to ensure that the survey can be performed without interruption, scheduling the survey is forbidden. If a dump is needed while observing the region, pause science and execute the dump on current exposure completion.
4. Ground-issued realtime commands with an attached time window will be scheduled by the FSW within that window, so as, if possible, not to interfere with ongoing activities. On completion of the ongoing activity, the realtime command is executed. If the ongoing activity will not complete by the end of the window, the realtime command is executed by the FSW no later than a database-specified time period prior to the expiration of the window. If the duration of the window is zero seconds, the realtime command is executed immediately upon receipt. If a realtime command must be executed prior to completion of the ongoing activity, the procedure for interruption of the activity and its subsequent treatment is specified by the onboard smart fault detection, diagnosis, isolation, and correction (SFDDIC) Agent.
5. Sun angle constraints may not be violated, either when slewing to a target or when observing a target. If any target within a cluster is in violation of this constraint, all targets within the cluster are considered to be in violation. If any part of a region is in violation of this constraint, the entire region is considered to be in violation.
6. A constraint on executing science observations is that SI calibrations must meet observation-specific accuracy requirements prior to initiation of the observation.

Relative to the previous assumptions, the following is a characteristic example of how FSW processing would function in performance of typical daily activities. Having completed the last science observation in the current cluster, the scheduling Agent requests that the SFDDIC Agent validate the remaining clusters relative to observing constraints. SFDDIC reports back that all remaining clusters are valid. Scheduling then asks the data monitoring and

trending Agent to identify those clusters that can be observed successfully given the current state of SI calibration. Monitoring and trending reports back that current calibration accuracy is insufficient to support successful observations at the nearest cluster, but is satisfactory at the remaining clusters. Planning and scheduling determines that the ground-specified priority attached to the nearest cluster is not high enough to justify scheduling an SI calibration update at this time, and instead directs that the attitude control Agent generate appropriate commanding to produce a slew to the first target in the next nearest cluster.

To effect a slew to the next target requires reaction wheel commanding, so the attitude control Agent's commands must pass through an executive Agent that interfaces with the FSW backbone. The backbone accepts the slew directive from the Agent community and interfaces with the reaction wheels to effect the slew. Following successful completion of the slew (as determined by the SFDDIC Agent), the ACS software in the backbone facilitates entry into fine-pointing mode by activating the quaternion star trackers (QSTs) and fine error sensor (FES). Once the necessary QST and FES data is available, the fine attitude determination Agent begins computing high accuracy attitude products. The data is simply stored in a file associated with the observation (managed by the SI data-storage Agent) and may be accessed by all users requiring the information. At the same time, and regularly before arrival at the target, the orbit determination Agent has produced a steady stream (once a second) of spacecraft, Solar position and velocity vectors, and trends position and velocity vectors, again in support of applications needing the information.

In particular, the attitude control Agent uses both the high accuracy attitude and orbit data to generate high precision attitude control commands that, again, are passed through the executive Agent to the FSW backbone, which in turn interfaces with the reaction wheels (after the backbone Quality Assures (QAs) the commands) to produce the desired pointing performance. Having established a stable platform at the target attitude, the planning and scheduling Agent directs the acquisition of the science target by the SI and initiation of the science observation specified by the ground. To this end, it notifies the SI commanding-and-configuration Agent to effect the necessary SI adjustments required to perform the desired science activity. The SI commanding-and-configuration Agent generates the associated commanding and forwards it to the executive Agent, which communicates the hardware changes to the FSW backbone so it can directly command the SI. All data output from the SI, whether from the target acquisition or execution of the science observation itself, is stored in the science observation file.

Specialized processing of the first SI data is performed by the SI data processing Agent to support target acquisition. Once the target has been acquired successfully, the activity proceeds to the science observation itself, which is processed onboard to support compact packaging prior to downlink. For example, lossless compression will be performed, and possibly processing to reject errors caused by cosmic rays. As the observation continues, the SI

data-storage Agent will progressively build the observation file so that, by the end of the activity, a complete, coherent record of the observation will have been compiled for downlink as a unified file under the authority of the SI data-storage and communications Agent.

When "data-take" at this target has been completed, the planning and scheduling Agent determines that a survey observation can be performed conveniently and schedules the necessary slew to the target. Processing flow then proceeds as described above for the earlier target, except following completion of data collection from the survey activity, the SI data processing Agent processes the survey data and identifies several point-targets of interest.

These targets are reported to planning and scheduling, which then adds them to the target list for immediate revisits, with observations to be performed according to canned, ground-specified scripts. The targets are then visited in an order defined by the planning and scheduling Agent. At the end of this activity, the SFDDIC (using data from the look-ahead modeling Agent) determines that a momentum dump is required. Planning and scheduling is notified, which decides that the dump should be performed now, and issues the necessary directive to the orbit maneuvering Agent, which also handles the thruster commanding for momentum reduction. As with the attitude control Agent, the orbit maneuvering Agent must forward its thruster commands to the executive Agent, which in turn relays them to the FSW backbone, which then performs its QA and communicates directly to the thrusters to cause the thrusters to fire in the manner specified. Finally, by this time the ground station antenna has become visible to the spacecraft (or *vice-versa*, depending on one's point of reference), and an electronic handshake between the ground station's and spacecraft's communications Agents is established. The handshake is initiated by the ground station Agent, but downlink of the science data onboard, including the recently built ground-specified and opportunistic survey files, is managed by the onboard communications Agent. The lights-out ground system autonomously validates each file as it is downlinked. As transmission of a file is deemed to be successful, the ground system notifies the onboard communications Agent and SI data-storage Agent that the onboard addresses associated with that data are free to be overwritten.

This completes the narrative illustrating the mutual communication and interaction of FSW subsystem Agents (along with some interaction with ground system Agents) in nominal performance of inflight activities of a typical science mission. In reality, the description provided is highly oversimplified: the communication flow in the example is quite sequential, whereas in reality there will be many parallel conversations in progress at any given time. Also, the assumptions specify a model of a far lower level of complexity than would be characteristic of a real mission. So the example provided should be viewed as simply a token of what would obtain in an actual application.

B.2 Space-to-Ground Dialogue Scenario

The dialogue in this scenario is initiated by the spacecraft and driven by the following assumptions:

1. The mission type is a 1-AU (i.e., 1 *astronomical unit*) drift-orbit survey mission.
2. The mission goal is to map out the entire celestial sphere to chart the microwave structure of space. Each mapping will take 6 months, so four mappings will be performed during the 2 year mission lifetime.
3. As the onboard antenna size is quite small and transmitting power is highly limited, the Deep Space Network (DSN) must be used for data capture. To reduce transmission costs by reducing downlink volume, the spacecraft will process all raw science data onboard and only downlink science end-products. The spacecraft will utilize beacon mode and will burst-transmit its processed science data on a low priority basis.
4. In the event of major anomalies that the autonomous SFDDIC Agent cannot handle, the spacecraft will notify the ground, downlinking a diagnostic file whose contents characterize the problems encountered.

Relative to these assumptions, consider this scenario for space-ground communications. As the survey work continues, the SI data processing Agent processes SI output and forwards the end-product to the SI data-storage Agent. In addition, raw SI data is stored in buffers (a precaution against the event of an SI anomaly). When the SFDDIC Agent (in conjunction with the data monitoring and trending Agent) validates a given subset of data and declares it to be acceptable, and also verifies nominal SI performance during that time period, the storage locations associated with the raw SI data are designated as available to be over-written.

When sufficient processed survey data has accumulated to warrant scheduling a downlink, the SI data-storage and communications Agent forwards to the executive Agent a request to turn on the transmitter to contact the DSN and request a downlink opportunity. This request is then relayed to the FSW backbone, which activates the transmitter, establishing contact with DSN. The DSN informs the spacecraft of its telemetry window. After the start of the window, the onboard Agent then downlinks all available, validated SI end-products. The lights-out ground system automatically verifies that all data received from the spacecraft in this pass are intact (i.e., have not been corrupted in transmission) and notifies the onboard SI data-storage Agent that the memory areas used for storage of the telemetered science data may now be overwritten.

Sometime later, after this conversation has terminated, the spacecraft loses (at least temporarily) one of its four reaction wheels. The FSW backbone responds by transitioning both the platform and payload to safemode. While in safemode, the SFDDIC (in conjunction with data monitoring and trending, as well as look-ahead modeling) evaluates the situation, both with respect to

the failed component and the overall spacecraft state. SFDDIC concludes that the spacecraft must remain in safemode pending consultation with ground resources, and establishes an emergency communications link with the ground via DSN using the procedure already discussed above.

Once a link is established, the SI data-storage and communications Agent dumps all recent data stored onboard to the ground system. The Agent also provides the results of the SFDDIC Agent's analysis as a starting point for the ground system's more definitive trouble shooting, to be conducted by an integrated team consisting of senior system engineers supported by the ground system's intelligent software Agents. As the ground system's work proceeds, requests for additional data from the spacecraft may be made via more regular and frequent DSN contacts. As new ideas are developed and need to be experimented with onboard, the onboard Agents may well join the ground system team and participate in a more active fashion until the problem is resolved and nominal function is restored.

This completes the narrative illustrating space-to-ground dialogues initiated by the flight system in nominal performance of typical inflight activities. In reality, the description provided is somewhat oversimplified, as the communication flow in the example is sparse and sequential, whereas in reality communications will probably be more frequent and there may be parallel conversations in progress during a single contact. Also, the assumptions specify a model of far less complexity than that characteristic of a real mission. So the example provided should be viewed as simply a token of what would obtain in an actual application.

B.3 Ground-to-Space Dialogue Scenario

The interaction in this scenario is initiated by the ground station. Consider a ground-space Agent dialogue driven by the following assumptions:

1. The mission type is LEO Earth-pointer.
2. The spacecraft determines its own orbit via GPS. Orbit stationkeeping maneuvers are performed autonomously onboard.
3. The mission goal is to observe all ground-specified targets while minimizing fuel expenditure so as to maximize mission lifetime.
   a) The spacecraft is provided with Earth coordinates of targets desired to be observed. Repeated observations are performed every 16 days. The spacecraft autonomously determines when the targets may be viewed during a 16-day cycle.
   b) The ground maintains onboard a set of observing scenario templates. Each target will be observed using one of those templates. The templates are populated by ground-alterable parameters controlling the observing process.
4. Targets are acquired autonomously by the spacecraft.

a) The spacecraft utilizes "quick-look" realtime image data to verify targete acquisition and to determine whether targets are obscured by cloud cover such that data collection is useless during this pass.
b) The spacecraft uses the results of its analysis of quick-look data to transition SI configuration autonomously to nominal high data rate mode should target conditions be suitable for the science observation to commence.
c) Pattern recognition is performed onboard, as necessary, to support the observation.

5. The spacecraft autonomously generates H&S commanding where necessary (e.g., SI re-configuration at SAA entrance/exit).

Each day for each ground station pass, the ground initiates contact with the spacecraft for the purpose of receiving science and engineering/diagnostic data. During ground-selected passes, the ground system uplinks to the spacecraft an updated target list, as well as changes to parameters controlling the science observations.

Each orbit, the planning and scheduling Agent uses data supplied by the orbit determination Agent to identify which ground targets can be observed and when. At a database-specified lead-time prior to encountering the target, planning and scheduling notifies the SI commanding-and-configuration Agent of the need to configure the SI for use according to the state specified by the template. SI commanding and configuration then generates the appropriate SI commanding and forwards its requests to the executive Agent, which relays the package to the FSW backbone, which (following its own command validation) ships the commands to the SI.

When a target is encountered, the SI data processing Agent examines the initial quick-look data to verify that the observation can be performed. The Agent passes its assessment to SI commanding and configuration and, if the conditions are suitable, commanding and configuration issues the necessary directives to initiate generation of high volume data. The directives are then relayed as before to the backbone so that the required adjustments can be made. Similarly, if special fine SI adjustments are needed to home-in on a specific landmark or feature, SI data processing performs the necessary pattern recognition function and informs SI commanding and configuration of its results. As science data is output from the SI, an observation file is constructed by the SI data-storage and communication Agent for downlink when requested by the ground system.

While these observations are performed, parallel onboard processing (controlled by the data-monitoring and trending Agent in conjunction with SFDDIC) determines when the orbit requires correction and informs the planning and scheduling Agent, which in turn schedules an orbit stationkeeping maneuver (commanding for which is generated by the orbit-maneuvering Agent at the request of planning and scheduling) so as not to conflict with upcoming science data takes. The same Agents involved in the orbit correction evaluation

also determine when SI reconfigurations need to be performed in response to SAA events. In both cases, agent-generated thruster and SI commands must be forwarded to the executive Agent and relayed to the backbone, which QAs the commands and issues the commands directly to the appropriate hardware.

Finally, any windowed realtime commands issued by the ground system are processed by the planning and scheduling Agent and inserted into the timeline of onboard commands/activities as necessary. Once scheduled, these realtime commands are treated onboard the same as any commands internally generated.

This completes the narrative illustrating ground-to-space dialogues initiated by the ground system in nominal performance of typical inflight activities. In reality, the description provided is somewhat oversimplified, as the communication flow in the example is sparse and sequential, whereas, in reality, communications will be more frequent and there will be parallel conversations in progress during a single contact. Also, the assumptions specify a model of far lower complexity than that characteristic of a real mission. So the example provided should be viewed as simply a token of what would obtain in an actual application.

B.4 Spacecraft Constellation Interactions Scenario

Whereas the scenario discussions above were restricted to cooperative efforts between Remote Agents on a single spacecraft or with counterpart Agents in the ground system, this subsection examines the far more elaborate topic of integrating the efforts of multiple teams of Agents on several spacecraft (as well as the ground). This higher level of complexity introduces a whole new set of issues unique to constellation work, including the following:

1. Is a moderating Agent/entity required to facilitate and referee dialogues among the members of a constellation?
2. Although all members of a constellation need to be aware of the results from a constellation dialogue, how do you decide which members should be direct participants in a given dialogue, and "who" makes that decision?
3. How are dialogues created, i.e., how are the topics for a dialogue selected? For example, does a constellation member with a "problem" simply call a "town meeting", does it submit its problem to a moderator for consideration, do problems un-resolvable by a single member get referred to the ground system to be dealt with, etc.?
4. As a dialogue progresses, can constellation members "casually" drop in and drop out? If so, is an ongoing record maintained as the dialogue proceeds so newly arriving or returning members can quickly get back up to speed? If so, how is the record maintained and by whom?
5. Are all dialogues short-term things with well-defined starts and endings, or can a dialogue extend over a long time duration, with gaps in activity

interspersed among periods of very high activity? In other words, is there the equivalent of regularly scheduled meetings?
6. Must all dialogues be originated from a well-defined issue or concern, or can some arise from general topics of interest, for example as a mechanism for developing and/or sharing knowledge among the various constellation members? Note the relationship of this question to question #5.
7. How does a new constellation member join the community of Agent groups? A new spacecraft may have improved methodologies or algorithms useful to the entire constellation. Would an Agent dialogue be initiated so it can share its new knowledge with the entire constellation?
8. Since the existing constellation members may "learn" new things in the course of their day-to-day activities, how do they share this knowledge with the other members so it becomes generally available? Prior to its absorption by the rest of the constellation, would other members be responsible for validating or QAing that new knowledge?
9. Once the knowledge base of the constellation grows beyond its original as-launched confines, how does this new "school-of-hard-knocks" knowledge get passed on to new members?
10. How does the constellation resolve differences between the new knowledge carried by new members vs. the empirical experience gained inflight by the old members?
11. Could different members of the constellation take on special roles to achieve general constellation goals? For example, could a subset of constellation members (equipped with higher capacity flight computers) conduct simulations or long-term studies of interest to the constellation as a whole?
12. For a communications satellite constellation, one can assume all members of the constellation will be at least compatible, if not identical. So interface incompatibility among members should not be a problem. However, suppose in the future it is desirable to form at least a temporary constellation of science satellites to engage in an observation campaign to study a celestial object or phenomenon of great scientific import. By what means could this goal be enabled or facilitated through the use of Remote Agents, and what special interface/architecture issues obtain?

The full impact of these issues for constellations is beyond the scope of this book. Here we will simply consider a high-level (and perhaps overly simplified) scenario for constellation member interactions purely for the purpose of illustrating how constellation behavior might be supported by the proposed FSW design. To this end, assume the following is the case:

1. The constellation consists of a set of 16 LEO Sun-synchronous and 4 GEO weather satellites (the fourth is a spare).
2. The OBCs of the larger GEOs have oversized processing power to enable transfer of LEO overflow processing, conduct background long-term studies and simulations, and manage overall direction of constellation behavior.

3. The GEOs rarely are replaced. The small LEOs are replaced fairly frequently due to orbit deterioration (to reduce LEO costs, it is assumed that onboard propulsion capacity is weak and fuel limits lifetime to 2 years).
4. The ground stations only communicate with GEOs. The GEOs communicate among themselves, the ground stations, and the LEOs. The LEOs only communicate with GEOs.
5. The LEOs are fairly primitive, and can be viewed somewhat simplistically as only possessing a FSW backbone. The GEOs possess the full range of Remote Agent functionality discussed previously. The ground system also is equipped with autonomous Agents for lights-out operation.
6. In support of the higher-level constellation goals, the LEOs' jobs are to pass their SI data to the GEOs and accept commanding and changes in mission objectives from the GEOs. Otherwise, the LEOs operate in a purely local manner, little different from the behavior of a ground-controlled single LEO. They collect their science data, perform necessary housekeeping functions (largely in response to external directives from the GEOs), and conduct very simple FDC functions.
7. The GEOs' job is to collect SI data from their equatorial orbits, communicate with the ground via the ground stations with which they are in permanent contact, and run the constellation. Running the constellation includes managing the collection of science data (including all LEO commanding), monitoring and trending all H&S data from the LEOs and GEOs, performing relatively short-term Continuing Process Improvement (CPI) type analytical studies to increase operational efficiency, and employing SFDDIC Agents to deal with some anomalies that cannot be handled optimally by FDC in the FSW backbones of the LEOs and GEOs.

The ground's job is to receive and archive all science data generated by the constellation, perform long-term analytical studies to increase operational efficiency, and support the GEOs in dealing with major inflight anomalies or failures.

A typical operational scenario is now presented, where some of the Agent interactions within a single spacecraft are glossed over in favor of a cleaner description of spacecraft-spacecraft dialogues. At the start of this scenario, the ground has just generated calibration updates for several of the SIs and updated SI observation templates for the LEO and GEO spacecraft. These data are uplinked to the GEOs with instructions to implement the updates as soon as possible without interfering with ongoing observations by the individual spacecraft, while maintaining consistency (as much as possible) among observations by those spacecraft. The planning and scheduling Agents of the three GEO spacecraft caucus (by way of a "teleconference" established by their communications Agents) and examine their anticipated processor loading (in consultation with their monitoring and trending and look-ahead modeling

Agents) over the next few hours. In this case, they determine that none of the three GEOs on their own can easily accommodate this planning and scheduling assignment within the high priority time demands specified by the ground without impact to their own science responsibilities. Rather than attempt to segment the planning and scheduling work, parcel out pieces to the individual GEOs, and assign one of the three GEOs to coordinate the effort, they decide instead to utilize the idle processing power of the fourth (spare) GEO and assign the job to GEO-4, leaving to GEO-1 responsibility for interfacing with GEO-4 when its task is completed.

While GEO-4 is performing the new intermediate-term planning and scheduling task, the other GEOs concentrate on their immediate routine jobs, namely receiving science data from LEOs for relaying to the ground and performing their own science assignments (and communicating the results to the ground) according to their current operating instructions.

For the first duty, the GEOs function in a manner quite similar to the traditional ground station, which has knowledge of the time and angle at which to view any given LEO as it comes into view over the horizon, thus starting a "view period" during which communications can take place. For the GEOs to perform their similar communications function with the LEOs, switching curves (in latitude and longitude) are maintained onboard the GEOs for use by the planning and scheduling Agents in conjunction with the look-ahead modeling and data-monitoring and trending Agents. The switching curves divide the "sky" into three 120-degree slices (where, it may be noted, none of the GEOs will be able to "see" the LEOs near the north and south poles). The curves defining the segment boundaries are padded so that when a LEO enters a padded region, preparations to initiate communications with the new GEO are begun and completed before the LEO leaves the padded region. To effect change in control, the GEO currently directing the LEO's actions instructs (via its communications Agent) the LEO to terminate its current telemetry link and reorient its main antenna towards the new GEO. The new GEO then initiates a link with the LEO and requests that flow of completed science products be renewed. Data received from LEOs are then formatted by the individual GEOs in observation files by the SI data-storage and communications Agents and are relayed to a designated single GEO (say, GEO-3) for integration into an overall global picture/assessment. As the GEOs themselves conduct their own science observations (as discussed in a previous section), the data from GEO-1 and GEO-2 are relayed to GEO-3 for merger with the LEO data.

When GEO-3 has completed the integration process, it also performs the data reduction processing required to convert the raw measurements into science end products. These results are then transmitted to the lights-out ground station for archiving and dissemination. Optionally, the raw measurements themselves may be transmitted to the ground for archiving. Note that GEO-3 does not integrate and process all the data all the time. Once GEO-3 begins its integration job, another of the three GEOs (say GEO-2) will be the col-

lection point for new data as it is generated by the constellation. So at any given time, one GEO will be collecting science data from the constellation, one GEO will be integrating and processing data, and a third GEO will be available for planning and scheduling work, which in this case was assigned to GEO-4, leaving GEO-1 available to respond to other ground requests as well as communications from GEO-4.

Once GEO-4 has completed its planning and scheduling work, it communicates its results to GEO-1. GEO-1 then passes the plan for installing the new calibrations and procedures to the other GEOs. Each GEO then instructs those LEOs under its control as to the changes that should be made. The time at which those changes should be made is also specified to ensure that when the next mass of data from the whole constellation is integrated, all data in the mix will have been generated, ideally, using the same calibrations and procedures. If that ideal condition is not possible, then any data obtained using old calibrations/procedures will, when contributed to integration, be so identified in the data/results transmitted to the ground.

B.5 Agent-based Satellite Constellation Control Scenario

Consider the scenario when one has many satellites with different viewing capabilities (IR, visible, or UV) orbiting a planet and one desires a full spectrum sweep of a certain portion of the planet. Traditionally, science team members and human controllers would need to identify the satellites with each different capability that will be making a pass over the section of the planet indicated. Human controllers would need to form a detailed, possibly quite intricate plan for the needed observations, and would need to organize a series of requests addressed to the satellites to perform the sweep, with all relevant details down to the transmission of the data back to earth. This type of activity entails inefficiencies and represents a questionable or wasteful use of manpower.

Through the use of an agent community that hierarchically and intelligently parses instructions, this could be done much more efficiently. Ideally, the human operator needs only to transmit a command similar to "Take a full spectrum picture of the area bounded by given latitude and longitude data and transmit the picture back in two days". An executive level satellite could receive this information, decompose it, and then negotiate with the agent community (where each satellite in orbit is part of the community) to attempt to schedule a plan. From there, each satellite could respond with information such as "will be passing over the site in 36 hours, I can take the picture in IR" or "will not be passing over the site for another 96 hours, I can not take the picture." Certain constraints may come into play also; for example, UV and visible light sensors are only useful when it is not night time at the given site. Responses of this nature may be similar to "will be passing over the site in 4 hours, but the site is currently on the dark side of the planet" or "will be passing over in 4 hours when site is on dark side, but will pass over again

in 20 hours when it is local noon." Of course, there are certain requests that just cannot be fulfilled. It is the executive's job to notice these, come up with the "closest fit" to the request issued from the human controller and report back with the closest fit to ask for a go-ahead on that schedule.

When all of the planning has been performed through the negotiation, the executive satellite could issue the plan to all image gathering satellites. The satellites will receive their plans, and internally they will schedule their own control (perhaps via an internal agent network for subsystem control) for setting up their imaging systems, recording the image, and transmitting it. The satellite will pass over the section of the planet in due course, record the images, and transmit them back to the executive satellite. The executive satellite will assemble the images when the whole spectrum has been covered, and transmit them back to the human controller at the next appropriate opportunity.

This scenario is a perfect illustration of a negotiating agent network. Although it is hypothetical, it unifies the concepts of hierarchical parsing networks between spacecraft, and within spacecraft.

B.6 Scenario Issues

One aspect of the presented design concepts that could create problems in flight is the layering of communications engendered by the FSW backbone. Commands, status messages, and data often must go through several checkpoints before they reach their intended hardware destination in order to guarantee platform and/or payload H&S, no matter what anomaly or failure conditions may obtain with the Remote Agents. There are two potential problems arising from this security paradigm. First, some commanding has associated with it very severe timing requirements. For example, in the case of HST, in order to satisfy its pointing accuracy and stability requirements, reaction wheel commands must be executed within 7 milliseconds of receipt of the gyro data from which the commands are derived. For this case, a data latency problem engendered from the time delays in relaying commands to the reaction wheels potentially could jeopardize meeting a fundamental mission requirement, unless the OBC and bus infrastructure are adequate to support the design. Second, the multiplication of messages and commands, especially if receipt of one always triggers an acknowledgement, could create a blizzard of traffic on the bus (or busses), leading to loss of information or even processing lock-up, again, unless the OBC and bus infrastructure are adequate to support the design. In other words, the presented design concepts, involving communications-and-computation-intensive negotiation processes between multiple Agents, imply the need for research into new spacecraft architectures and the crucial need for certain minimum levels of performance of future onboard communications and computing resources.

C

Acronyms

AC - Autonomic Computing
ACE - Attitude Control Electronics
ACL - Agent Communication Language
ACS - Attitude Control Subsystem
ACT - Agent Concepts Testbed
AE - Autonomic Element
AFLOAT - an Agent-based Flight Operations Associate
AI - Artificial Intelligence
AIFA - Archive Interface Agent
AM - Autonomic Manager
AMS - Active Middleware Service
ANS - Autonomic Nervous System
ANTS - Autonomous Nano Technology Swarm
AOS - Acquisition of Signal
ARPA - Advanced Research Projects Agency
ASM - All Sky Monitor
BAT - Burst Alert Telescope
BN - Bayesian Networks
C&DH - Command and Data Handling
CBR - Case Based Reasoning
CCC - Constellation Control Center
CCD - Charge-Coupled Device
CGRO - Compton Gamma Ray Observatory
CIM - Common Information Model
CLIPS - C-Language Integrated Production System
CMA - Contact Manager Agent
COTS - Commercial Off the Shelf
CPU - Central Processing Unit
CSS - Coarse Sun Sensor
DARPA - Defense Advanced Research Projects Agency
DBIFA - Database Interface Agent

DIS - Distributed Interactive Simulation
DS - Deep Space
DS1 - Deep Space 1
DSN - Deep Space Network
DSS - Digital Sun Sensor
EO-1 - Earth Observing - 1
EOS - Earth Observing System
EP - Explorer Platform
EUVE - Extreme Ultraviolet Explorer
FAST - Formal Approaches to Swarm Technologies
FDC - Fault Detection and Correction
FES - Fine Error Sensor
FGS - Fine Guidance Sensor
FIPA - Foundations of Intelligent Physical Agents
FIRE - Fault Isolation and Resolution Expert
FOT - Flight Operations Team
FOV - Field of View
FSS - Fine Sun Sensor
FSW - Flight Software
FTC - Fault Tolerant Computing
GCI - Geocentric Inertial
GEO - Geosynchronous Earth Orbit
GIFA - GenSAA/Genie Interface Agent
GN&C - Guidance, Navigation, and Control
GPM - Global Precipitation Measurement mission
GPS - Global Positioning System
GRB - Gamma Ray Burst
GSFC - Goddard Space Flight Center
GSS - Ground Station Simulator or Generalized Support Software or Ground Support System or Ground Support Software
GTDS - Goddard Trajectory Determination System
H/W - Hardware
H&S - Health and Safety
HBM - Heart-Beat Monitor
HEAO - High Energy Astronomical Observatory
HGA - High Gain Antenna
HLA - High Level Architecture
HST - Hubble Space Telescope
I/O - Input/Output
IBM - International Business Machines
ICT - Information and Communications Technology
IMU - Inertial Measurement Unit
IR - Infrared
IRU - Inertial Reference Unit
ISA - Interface Services Agent

IT - Information Technology
IUE - International Ultraviolet Explorer
JPL - Jet Propulsion Lab
JWST - James Webb Space Telescope
KIF - Knowledge Interchange Format
KQML - Knowledge Query and Manipulation Language
KSE - Knowledge Sharing Effort
KX - Kinesthetics eXtreme
LARA - Lander Amorphous Rover Antenna
LEO - Low Earth Orbit
LISA - Laser Interferometer Space Antenna
LOG - Log agent
LOS - Loss of Signal
MA - Multiple Access
MAP - Microwave Anisotropy Probe
MAPE - Monitor, Analyze, Plan and Execute
MAS - Multi-Agent System
MC - Managed Component
MIDEX - Medium-class Explorer
MIFA - MOPSS Interface Agent
MMA - Mission Manager Agent
MMS - Magnetospheric MultiScale
MOCC - Mission Operations Control Center
MOPSS - Mission Operations Planning and Scheduling System
MTB - Magnetic Torquer Bar
NASA - National Aeronautics and Space Administration
NFI - Narrow Field Instruments
NIR - Near-Infrared
NSSC - NASA Standard Spacecraft Computer
OAO - Orbiting Astronomical Observatory
OBC - Onboard Computer
OPE - Observation Plan Execution
OSO - Orbiting Solar Observatory
OTA - Optical Telescope Assembly
PAGER - Pager Interface Agent
PAM - Prospecting Asteroid Mission
PBM - Pulse-Beat Monitor
PCA - Proportional Counter Array
PCS - Pointing Control Subsystem
PID - Proportional-Integral-Derivative
PSA - Planner/Scheduler Agent
QA - Quality Assurance
QST - Quaternion Star Tracker
R&D - Research and Development
RF - Radio Frequency

RSDO - Rapid Spacecraft Development Office
RTS - Relative Time Sequence
RXTE - Rossi X-ray Timing Explorer
SA - Solar Array
SAA - South Atlantic Anomaly
SAC - Situated and Autonomic Communications
SAMPEX - Solar Anomalous and Magnetospheric Particle Explorer
SARA - Saturn Autonomous Ring Array
SC - Spacecraft
SDO - Solar Dynamics Observatory
SFDDIC - Smart Fault Detection, Diagnosis, Isolation, & Correction
SI - Science Instrument
SMEX - Small Explorer
SMM - Solar Maximum Mission
SMP - Statistics Monitor Program
SSA - S-band Single Access
SSA - System Services Agent
SSR - Solid State Recorder
SysMMA - System Monitoring and Management Agent
TAM - Three Axis Magnetometer
TBS - To Be Specified
TCO - Total Cost of Ownership
TDRS - Tracking and Data and Relay Satellite
TDRSS - TDRS System
TMON - Telemetry Monitor
TOO - Target of Opportunity
TRMM - Tropical Rainfall Measuring Mission
TSM - Telemetry & Statistics Monitor
TWINS - Two Wide-angle Imaging Neutral-atom Spectrometers
UARS - Upper Atmosphere Research Satellite
UI - User Interface
UIA - User Interface Agent
UIFA - User Interface Agent
UV - Ultraviolet
UVOT - UV/Optical Telescopoe
VIFA - VisAGE Interface Agent
WMAP - Willsinson Microwave Anisotropy Probe
XRT - X-ray Telescope

D

Glossary

Angular Momentum Dump - See Momentum Dump.

Attitude - The orientation of the spacecraft in inertial space. Usually, attitude defines the orientation of all three spacecraft axes with respect to an inertial reference frame, although for spin-stabilized spacecraft it often is the case that only the orientation of the spin axis is specified.

Attitude Actuator - A control hardware component that generates control torques required to maintain spacecraft stability, null attitude errors, reorient the spacecraft to a new orientation with respect to an inertial reference frame, etc.

Attitude Control - The mechanism for establishing and maintaining a desired spacecraft orientation with respect to an inertial reference frame.

Attitude Control Accuracy - A quantitative measurement of the error in maintaining the spacecraft attitude at its desired orientation with respect to an inertial reference frame.

Attitude Control Torque - Torques intentionally applied to the spacecraft to maintain or establish a desired spacecraft orientation with respect to an inertial reference frame.

Attitude Determination - The computation of the spacecraft orientation relative to a specified reference frame. For celestial-pointing spacecraft, the reference frame is usually either the geocentric inertial (GCI) frame or the heliocentric inertial frame.

Attitude Determination Accuracy - A quantitative measurement of the error in the computed spacecraft attitude.

Attitude Dynamics - The study of a spacecraft's motion about its center of mass.

Attitude Maneuver - A commanded change in the spacecraft's desired attitude, as opposed to a torque applied to null errors in the actual attitude relative to the desired attitude. A large attitude maneuver is called a *slew*.

Attitude Matrix - A specification of the spacecraft orientation in direction cosine matrix format. The direction cosine matrix is a 3x3 square matrix. The nine elements of the matrix are the cosines of the angles between the

three spacecraft-body unit vectors and the three reference axes relative to which the spacecraft orientation is defined.

Attitude Quaternion - A specification of the spacecraft orientation in quaternion format.

Attitude Sensors - Spacecraft hardware and electronics providing measurement data that can be used to determine the spacecraft orientation or changes in the spacecraft orientation.

Autonomic - Of or pertaining to the capacity of a system to control its own internal state and operational condition.

Autonomic Communications - A research field with the same motivators as the Autonomic Computing concept with particular focus on the communications research and development community (see SAC).

Autonomic Computing - Overarching initiative to create self-managing computer-based systems, with a metaphor inspired by the biological autonomic nervous system.

Autonomic Element - An Autonomic Manager and a Managed Component considered together.

Autonomic Manager - A control loop and components to provide autonomic self-* for the managed component.

Autonomic Systems - Often used synonymously with Autonomic Computing; sometimes used as a synonym for autonomicity from a systems perspective, and sometimes used in the sense that $AutonomicSystems = AutonomicComputing + AutonomicCommunications$.

Autonomicity - The quality of having an autonomic capability.

Autonomy - A system's capacity to act according to its own goals, percepts, internal states, and knowledge, without outside intervention.

Celestial-pointer - A spacecraft whose fine-pointing science instruments are oriented towards "fixed" points on the celestial sphere. For example, a spacecraft that slews from one attitude to another to observe a series of X-ray point-sources would be a member of the class of celestial pointers.

Center of Mass - Average position of a system of objects, weighted in proportion to their masses.

Charge Coupled Device Star Tracker - A star tracker that detects stars by digitally scanning an array of photosensitive components (pixels). The star tracker integrates the electrical charge in the pixels "struck" by (for example) starlight. The measurement is performed by reading the pixel output line by line.

Coarse Sun Sensor - A device that measures photocell output as a function of Sun angle. Since the amount of energy hitting the photocell varies as the cosine of the Sun angle (normal incidence for null Sun angle, grazing incidence for 90 degree Sun angle), CSSs are also referred to as cosine detectors. Note that this scheme provides an analog representation of the Sun angle.

Command Quaternion - Desired spacecraft orientation, with respect to an inertial reference frame, expressed in quaternion format.

Commanded Attitude - Desired spacecraft orientation with respect to an inertial reference frame.

Constellation - Two or more spacecraft engaged in coordinated operations with the objective of meeting a set of mission requirements.

Control Loop - Closed loop of feedback control.

Control Torque - Torque generated by spacecraft actuators for the purpose of controlling the spacecraft's attitude.

Digital Sun Sensor - A device that measures output from a series of photocells to determine Sun angle. If a photocell's output is greater than a threshold, it is considered to be "on". The pattern of "on" photocells is directly associated with the Sun angle. Note that this scheme provides a digital representation of the Sun angle.

Distributed Satellite Systems - Multiple networked spacecraft, analogous to distributed client-server, or networked, computing systems of today, as opposed to the traditional "monolithic" centralized computing environment of the past.

Distributed Science Mission - Multiple networked space assets, analogous to distributed client-server, or networked, computing systems of today, as opposed to the traditional "monolithic" centralized computing environment of the past. The assets could include rovers, stationary instrument packages, and spacecraft.

Downlink - A point-to-point RF communications channel carrying data from the spacecraft to the ground.

Earth-Pointer - A spacecraft whose fine-pointing science instruments are oriented along the nadir vector towards the Earth.

Effector - In the context of autonomic management, a defined means to bring about a change to a part of the managed component.

Environment Awareness - A capability of a system through which the system is continually able to perceive its external operating conditions in relation to its knowledge of its abilities. From this perspective, environment awareness may be considered a part of self-awareness—the ability to know one's place in the environment. In another view of environment awareness, the environment is aware of the individuals themselves—for instance, through their heartbeat or pulse.

Ephemeris - In the context of space missions, a table of positions (of either celestial objects or spacecraft) at a given time (or at given times) in a given coordinate system. Also, colloquially referred to as an empirical (as opposed to algorithmic) specification of a spacecraft's orbital position (either historical or predicted).

Fine Sun Sensor - A high precision digital Sun sensor (DSS).

Formation Flying - A flight concept in which multiple spacecraft perform their science operations while keeping a fixed position relative to one another.

Geomagnetic Field - The magnetic field of the Earth.

Geostationary Orbit - An orbit around the Earth that maintains a spacecraft directly above the same point on the Earth's surface at all times. This is necessarily an equatorial orbit (with orbit inclination to the equatorial plane equal to zero degrees) where the orbit altitude is precisely chosen so that its associated period is 24 hours, matching the Earth's rotational rate.

Geosynchronous Orbit - An orbit having the same altitude as a geostationary orbit, but not necessarily maintaining the spacecraft above the same point on the Earth at all times. Geosynchronous orbits may have non-zero inclinations.

Gimbal - A rotatable or pivotable hardware contrivance whereby an attached element of a spacecraft, for example a dish antenna or solar array, may be reoriented (in two degrees of freedom) relative to the body of the spacecraft.

Global Positioning System - A constellation of low Earth orbiting satellites continually broadcasting radio signals supporting realtime, onboard determination of position by spacecraft with compatible receivers.

Goddard Trajectory Determination System - The primary ground software system at NASA Goddard Space Flight Center for computing definitive and predictive spacecraft ephemerides.

Gyro - Originally, a mechanical sensor containing a spinning mass that exploits conservation of angular momentum to measure changes in attitude. Recently, spacecraft have employed gyros that utilize non-mechanical structures (for example, hemispheric resonating gyros (HRGs) and laser ring gyros).

Gyro Drift - A ramping error (varying with time) in a gyro's output.

Gryo Scale Factor & Alignment Calibration - An operational process consisting of a series of large slews executed to generate gyro data that the ground system uses to determine the gyro scale factors and gyro alignments relative to absolute attitude sensors (such as star trackers).

Gyroscope - See Gyro.

HBM - Heart-Beat Monitor

Inertial Measurement Unit - See Gyro.

Inertial Reference Unit - See Gyro.

Kalman Filter - A sequential estimator with a fading memory commonly used for realtime spacecraft attitude estimation. Implemented in the ACS flight software of most GSFC spacecraft flying gyros. Enables onboard estimation of both attitude error and gyro drift bias.

Lagrange Point - Positions of stable or pseudo-stable gravitational equilibrium within a three-body system consisting of two major bodies and one body of negligible mass relative to the other two. In a gravitational three-body system, there are, relative to one of the two major bodies, always exactly five such points, referred to as L1, L2, L3, L4, and L5, all in the plane of the orbit of the one major body about the other. The three pseudo-stable points (L1, L2, and L3) lie along an axis between the

one major body and the other. The two stable points (L4 and L5) are off-axis, in positions leading and following the one major body on its orbital path around the other major body. An example of clumping of small natural objects at off-axis Lagrange Points is the Trojan Asteroids (here, the two large gravitational bodies are the Sun and Jupiter). See Figure 3.2 illustrating Lagrange Points.

Lagrange Point Orbit - The complex, non planar motion of a spacecraft when in orbit, the so-called *halo orbit*, near one of the unstable Lagrange Points, L1, L2, or L3. The halo orbit in general cannot be maintained without station-keeping.

Libration Point - See Lagrange Point.

Limit Checking - A validation procedure in which a data point value is checked against a threshold (upper, lower, or both).

Low Earth Orbit - An orbit whose altitude above the Earth's surface is between about 160 km and about 2000 km. Atmospheric drag increases dramatically at lower altitudes. A popular LEO altitude range is 500 to 600 km.

Magnetic Coil - A wire wrapped about a cylinder in a series of loops; a magnetic field results when the wire carries an electrical current. When the cylindrical space contains a ferromagnetic core, the configuration is called a *magnetic torquer bar (MTB)*.

Magnetic Disturbance Torque - A spacecraft torque generated by the interaction of the spacecraft's residual magnetic dipole as the spacecraft moves along its orbital path through the external magnetic field.

Magnetic Torque - Torque on the spacecraft arising from the interaction between fields generated by the spacecraft's magnetic coils and the external magnetic field. If the torque arises from an interaction between a spacecraft residual dipole moment and the external magnetic field, the torque is called the *magnetic disturbance torque*.

Magnetic Torquer Bar - See Magnetic Coil.

Magnetometer - An attitude sensor measuring the strength and direction of the magnetic field external to the spacecraft. Note that this measurement will include not only the geomagnetic field (for spacecraft in Earth orbit), but also contributions from MTBs and spacecraft residual dipole moment. Processing of the raw magnetometer measurements must remove these other field sources to obtain accurate geomagnetic field measurements.

Managed Component - A component that is protected by the Autonomic Manager.

MAPE - Monitor, Analyze, Plan and Execute components within Autonomic Manager.

Measured Attitude - The current best estimate of the spacecraft attitude.

Molniya Orbit - An orbit designed to support Russian communications satellites. It freezes the orbit's perigee over the Southern hemisphere, tilting the orbit to place the apogee over the Northern latitudes, ensuring

maximum communication opportunities between spacecraft and ground stations.

Momentum Dumping - A procedure, for spacecraft controlled by reaction or momentum wheels, whereby angular momentum is removed from the wheel system to prevent the wheels from saturating (a circumstance where wheel speeds can be run up to maximum values). Angular momentum dumping utilizes other attitude actuators to shift or remove angular momentum from the wheels (e.g., by applying current to magnetic coils, thereby generating a spacecraft counter-torque by means of coupling to the geomagnetic field).

Momentum Wheel - A large wheel spinning at a relatively high rotation rate to provide gyroscopic stabilization for a spacecraft. Distinct from Reaction Wheels (q.v.), which are typically operated at relatively low rotation rates.

Nadir Vector - In the context of the gravitational field of a massive body, the nadir at a given point is the direction along the force of gravity at that point. For Earth-orbiting spacecraft, the vector directed towards the geocenter. See Zenith.

Nadir-Pointer - A spacecraft whose primary science instrument is directed along the nadir vector.

NanoSpacecraft - A small spacecraft characterized as having a weight of about 10 kg or less, a cylindrical diameter about 30 cm or less, and a low cost.

Networked Science Mission - See Distributed Science Mission.

Occultation - A geometric condition when the view of the target of a spacecraft sensor or science instrument is obstructed by a celestial body.

Orbit Acquisition - Achieving mission orbit.

Orbit Decay - For Earth-orbiting spacecraft, decrease in apogee due to atmospheric drag.

Orbit Determination - Computation of the orbit of a spacecraft (or celestial body) in inertial space.

Orbit Determination Accuracy - A quantitative measurement of the validity of the computed spacecraft orbit.

Orbit Dynamics - The study of the motion of the spacecraft's center of mass.

Orbit Elements - A set of parameters specifying the size, shape, and orientation of the orbit in inertial space, as well as the location of the spacecraft at a given time (the epoch time). An example is the osculating Keplerian elements.

Orbit Generator - An algorithm-based computational means of predicting the future position and velocity of body in orbit. Commonly, the algorithm can also be used to compute orbits into the past from a given set of orbital elements.

Orbit Maneuver - A commanded change in the spacecraft's orbit, accomplished by producing a net thrust force on the spacecraft, generally by

firing a thruster. If applied to null errors in the actual orbit relative to the desired orbit, the maneuver is called a *station-keeping maneuver*.

Orbit Normal - A vector perpendicular to the orbital plane. The normal vector obtained by the *right hand rule*, with the fingers curled in the direction of spacecraft orbital motion, is *positive orbit normal*. The opposite vector is *negative orbit normal*.

Orbit Propagation - Extrapolation of spacecraft position and velocity vectors from actual measurements at an earlier time.

Period - An orbital period is the time it takes a spacecraft (or celestial object) to complete an orbit.

Predicted Orbit - Extrapolated spacecraft position and velocity (as a function of time) determined by applying mathematical models of physical processes (e.g., the Earth's gravitational potential) to actual measurements of the spacecraft orbit at an earlier time.

Propagation - For orbits, extrapolation of a spacecraft position and velocity from a known starting value. For attitudes, use of relative attitude information (from gyros) to extrapolate spacecraft absolute pointing from a starting, measured absolute attitude (for example, from star tracker data).

Propellant - Thruster fuel.

Pulse-Beat Monitor - Extension of Heart-Beat Monitor with health urgency tones

Quaternion - A mathematical representational system involving parameterization of the three pieces of information supplied by Euler angles in terms of a four-dimensional unit vector, which facilitates descriptions of spacecraft attitudes. The quaternion lacks the singularity issues present in Euler angle formulations and is more compact than the nine elements of the direction cosine matrix. Quaternions also are easily manipulated to determine (or incorporate) changes in attitude, a useful feature when supporting attitude slews.

Reaction Wheel - A flywheel rotated at a relatively low rate with an electric motor, used on a spacecraft to transfer angular momentum to or from the spacecraft body and thereby effect a change in the spacecraft's attitude without firing a thruster. Distinct from Momentum Wheel (q.v.), which typically has a relatively high rotation rate.

Reference Data - Inertial frame information from a model, catalog, etc. that can be combined with attitude observations in order to determine the spacecraft orientation.

Self-* - Self-managing properties

Self-anticipating - The ability to predict likely outcomes or to simulate self-* actions.

Self-awareness - The ability to perceive and compute with its own internal state in relation to its own knowledge and capabilities. Relates to the concept of 'Know thy self'.

Self-chop - The initial four (and generic) self properties (Configuration, Healing, Optimization, and Protection)

Self-configuring - A system's ability to configure and re-configure itself to meet policies/goals.

Self-critical - The ability to consider whether policies are being met or goals are being achieved (see self-reflecting)

Self-defining - The ability to reference (and operate in a manner determined by) internal data and the internal definitions of that data (i.e., metadata). Primarily related to definitions of goals and policies (perhaps as derived from self-reflection).

Self-governing - The capability of operating autonomously and being responsible for achieving goals and performing tasks.

Self-healing - In the reactive sense, the capability of self-fixing faults; in the proactive sense, the capability of predicting and preventing faults.

Self-managing - The capability of operating autonomously and being responsible for wider self-* management.

Self-optimizing - A system's capability of dynamically optimizing its own operation.

Self-organizing - A system's capability of organizing its own efforts. Often used relative to networks and communications.

Self-protecting - A system's capability of protecting itself through perception of potential threats and prediction of outcomes of situations in the environment, and through self-configuring to minimize potential harm.

Self-reflecting - The capability of assessing routine and reflex operations of self-* operations and determining whether they are as expected. May involve self-simulation to test scenarios.

Self-simulation - The capability of generating and testing possible scenarios without affecting the live system.

Selfware - Self-managing software or firmware.

Sensor - In the context of autonomic capabilities, a means to measure a part of the managed component. In an ACS context, a measuring device whose output can be used to determine the spacecraft's attitude, either in absolute or relative terms.

Situated and Autonomic Communications - Local, self-managed information flows in reacting to environment and context changes. Refers to the communication & networking vision of being task- and knowledge-driven and fully scalable.

Slew - A large change in orientation, e.g., a large attitude maneuver by a spacecraft.

South Atlantic Anomaly - A region of space near the Earth over the south Atlantic ocean where the van Allen radiation belt makes its closest approach to the Earth's surface, with increased intensity of radiation.

Star Tracker - A star detecting device used for spacecraft attitude determination and control. Stars registering an intensity above a commanded threshold are detected and tracked until a break-track is ordered. While

the star is being tracked, the star tracker measures the location of the star in the field of view (FOV), as well as the star's magnitude. For early star trackers, the output was location and magnitude, but recently quaternion star trackers have been flown that output an attitude quaternion (relative to the tracker frame as opposed to the body frame) directly.

Station-Keeping Maneuver - A spacecraft orbital maneuver performed to null errors in the actual orbit relative to the desired orbit.

Stellar-pointer - A spacecraft that conducts its science by pointing a science instrument (or instruments) at targets on the celestial sphere, i.e., stars, galaxies, etc. Stellar-pointers are slewed to a commanded attitude, acquire their target(s), conduct their science, and then slew to the next target(s).

Sun Sensor - An attitude sensor that detects the presence of the Sun and, in the case of analog and digital Sun sensors, also outputs measurements from which the Sun vector or Sun angle can be computed.

Sun-Pointer - A spacecraft whose attitude is maintained so as to keep its science instrument(s) oriented towards the Sun.

Sun-Synchronous Orbit - A spacecraft orbit whose plane rotates at the same rate as the Earth's orbital rate about the Sun. The spacecraft orbital plane motion then matches the apparent motion of the Sun (relative to the Earth). Sun-synchronous spacecraft imaging the Earth will therefore (for example) always image points on the Earth's equator at the same local time.

Swarm - A large group of autonomous individuals each having simple capabilities, cooperative actions, and no global knowledge of the group's objective.

Synchronous Satellite - For Earth orbiting spacecraft, a spacecraft whose orbital motion matches the Earth's rotational motion. If the spacecraft inclination is zero, the spacecraft will appear to hover over the same point on the Earth's surface.

Tachometer - A device that measures the rotation rate of a reaction wheel or momentum wheel.

Three-Axis Attitude - A specification of the orientation of a spacecraft's body axes with respect to a reference frame, typically expressed as a transformation from the reference frame to the body frame.

Three-Axis Magnetometer - An attitude sensor that measures the magnitude and direction of the magnetic field in which the spacecraft is immersed.

Three-Axis Stabilized - A type of spacecraft the orientations of all of whose body axes are controlled; distinct from a spin-stabilized spacecraft, which only controls the orientation of its spin axis.

Thruster - Spacecraft hardware that generates thrust by expelling propellant.

Torque - The vector rate of change of the angular momentum. A rigid body experiences a torque if a net force is applied to the body along a line that

does not pass through the body's center of mass. Such forces introduce rotations about the object's center of mass.

Tracking and Data Relay Satellite - A communications relay satellite in NASA's TDRSS. TDRS satellites are geostationary spacecraft.

Tracking and Data Relay Satellite System - The collection of NASA's TDRS satellites, their ground stations, and their control systems.

Uplink - An RF communications channel for data flow from the ground to the spacecraft.

Virtual Mission - A mission that consists of both virtual and real spacecraft assets to meet a mission objective that would otherwise require one or more real spacecraft and their instruments to be designed, constructed, launched, and operated.

Virtual Platform - A "virtual spacecraft" whose "payload" is composed of the instrument(s) or payload(s) aboard two or more "real" spacecraft. By extension, the virtual payload may also include instruments located at ground-based observatories.

Zenith - In the context of the gravitational field of a massive body, the zenith at a given point is the direction opposite to the force of gravity at that point. See Nadir.

Zenith-Pointer - A spacecraft whose primary science instrument(s) is (are) directed "up" along the local vertical (i.e., along the zenith vector).

References

1. A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.
2. M. Aarup, K. H. Munch, J. Fuchs, R. Hartmann, and T. Baud. Distributed intelligence for ground space systems. In *Proc. Third International Symposium on Articial Intelligence, Robotics, and Automation for Space (I-SAIRAS 94)*, pages 67–70, Pasadena, California (USA), 18–20 April 1994. NASA Jet Propulsion Laboratory.
3. M. Agarwal, V. Bhat, H. Liu, V. Matossian, V. Putty, C. Schmidt, G. Ahang, L. Zhen, M. Parashar, B. Khargharia, and S. Hariri. AutoMate: Enabling autonomic applications on the grid. In *Proc. The Autonomic Computing Workshop, 5th Int. Workshop on Active Middleware Services (AMS'03)*, Seattle, Washington (USA), 25 June 2003.
4. S. Aiber, O. Etzion, and S. Wasserkrug. The utilization of AI techniques in the autonomic monitoring and optimization of business objectives. In *IJCAI Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems*, Acapulco (Mexico), 10 August 2003.
5. J. Allen. *Natural Language Understanding*. The Benjamin Cummings Publishing Company, Inc., 1994.
6. E. A. Alluisi. The development of technology for collective training: SIMNET, a case history. In L. D. Voss, editor, *A Revolution in Simulation: Distributed Interaction in the '90s and Beyond*. Pasha Publications, Arlington, Virginia (USA), 1991.
7. T. Ames and S. Henderson. The Workplace Distributed Processing Environment. In C. Hostetter, editor, *Proc. 1993 Goddard Conference on Space Applications of Artificial Intelligence*, pages 181–188, Greenbelt, Maryland (USA), 10–13 May 1993. NASA Goddard Space Flight Center, NASA Conference Publication 3200.
8. A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1), January-March 2004.
9. T. Balch, A. Feldman, and Z. Khan. Automatic classification of insect behavior using computer vision and behavior recognition. In *Proc. Second International*

Workshop on the Mathematics and Algorithms of Social Insects, Atlanta, Georgia (USA), 16–17 December 2003.
10. D. Bantz and D. Frank. Challenges in autonomic personal computing, with some new results in automatic configuration mangement. In *Proc. IEEE Workshop on Autonomic Computing Principles and Architectures (AUCOPA 2003)*, pages 451–456, Banff, Alberta (Canada), 22-23 August 2003.
11. D. F. Bantz, C. Bisdikian, D. Challener, J. P. Karidis, S. Mastrianni, A. Mohindra, D. G. Shea, and M. Vanover. Autonomic personal computing. *IBM Systems Journal*, 42(1):165–176, 2003.
12. G. Beni. The concept of cellular robotics. In *Proc. 1988 IEEE International Symposium on Intelligent Control*, pages 57–62. IEEE Computer Society Press, Los Alamitos, California (USA), 1988.
13. G. Beni and J. Want. Swarm intelligence. In *Proc. Seventh Annual Meeting of the Robotics Society of Japan*, pages 425–428, Tokyo (Japan), 1989. RSJ Press.
14. K. P. Birman, R. van Renesse, and W. Vogels. Navigating in the storm: Using Astrolabe for distributed self-configuration, monitoring and adaptation. In *The Autonomic Computing Workshop, 5th Int. Workshop on Active Middleware Services (AMS'03)*, pages 4–13, Seattle, Washington (USA), 25 June 2003.
15. D. G. Boden and W. J. Larson. *Cost-Effective Space Mission Operations*. McGraw-Hill, 1996.
16. E. Bonabeau and G. Théraulaz. Swarm smarts. *Scientific American*, pages 72–79, March 2000.
17. A. H. Bond and L. Gasser. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers, San Mateo, California (USA), 1988.
18. W. K. Braudaway and S. M. Harkrider. Implementation of the high level architecture into DIS-based legacy simulations. In *Proc. Spring 1997 Simulation Interoperability Workshop*, Orlando, Florida (USA), 1997.
19. A. Brown and D. Patterson. Embracing failure: A case for recovery-oriented computing. In *Proceedings of High Performance Transaction Processing Symposium*, October 2001.
20. A. B. Brown, J. Hellerstein, M. Hogstrom, T. Lau, S. Lightstone, P. Shum, and M. P. Yost. Benchmarking autonomic capabilities: Promises and pitfalls. In *Proc. ICAC'04: International Conference on Autonomic Computing*, pages 266–267, May 2004.
21. S. Carlson. Artificial life: Boids of a feather flock together. *Scientific American*, November 2000.
22. E. K. Casani, B. Wilson, and R. Ridenoure. The new millennium program: Positioning nasa for ambitious space and earth science missions for the 21st century. *AIP Conference Proceedings*, 361(3):1553–1558, 1996.
23. W. Cedeno and D. K. Agrafiotis. Combining particle swarms and k-nearest neighbors for the development of quantitative structure-activity relationships. *Int. J. Comput. Res.*, 11(4):443–452, 2003.
24. H. Chalupsky, T. Finin, R. Fritzson, D. McKay, S. Shapiro, and G. Weiderhold. An overview of KQML: A knowledge query and manipulation language. Technical report, KQML Advisory Group, Apr. 1992.
25. G. Cheliotis and C. Kenyon. Autonomic economics: A blueprint for self-managed systems. In *IJCAI Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems*, Acapulco (Mexico), 10 August 2003.

26. S. Chien, R. Sherwood, D. Tran, R. Castano, B. Cichy, A. Davies, G. Rabideau, N. Tang, M. Burl, D. Mandl, S. Frye, J. Hengemihle, J. D'Agostino, R. Bote, B. Trout, S. Shulman, S. Ungar, J. Van Gaasbeck, D. Boyer, M. Griffin, H. hua Burke, R. Greeley, T. Doggett, K. Williams, V. Baker, and J. Dohm. Autonomous science on the EO-1 mission. In *Proc. International Symposium on Artificial Intelligence Robotics and Automation in Space (i-SAIRAS)*, May 2003.
27. D. J. Clancy. NASA challenges in autonomic computing. In *The Second Almaden Institute*, IBM Almaden Research Center, San Jose, California (USA), 10–12 April 2002.
28. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski. A knowledge plane for the Internet. In *Proc. ACM SIGCOMM 2003: Applications, technologies, architectures, and protocols for computer communication*, 2003.
29. P. E. Clark, S. A. Curtis, and M. L. Rilee. ANTS: Applying a new paradigm to Lunar and planetary exploration. In *Proc. Solar System Remote Sensing Symposium*, Pittsburgh, Pennsylvania (USA), 20–21 September 2002.
30. S. Curtis, M. Rilee, W. Truszkowski, C. Cheung, and P. Clark. Neural basis function control of super micro autonomous reconfigurable technology (SMART) nano-systems. In *Proc. First AIAA Intelligent Systems Technical Conference*. AIAA, 20–22 September 2004.
31. S. A. Curtis, J. Mica, J. Nuth, G. Marr, M. L. Rilee, and M. K. Bhat. ANTS (Autonomous Nano-Technology Swarm): An artificial intelligence approach to Asteroid Belt resource exploration. In *Proc. Int'l Astronautical Federation, 51st Congress*, October 2000.
32. S. A. Curtis, W. F. Truszkowski, M. L. Rilee, and P. E. Clark. ANTS for the human exploration and development of space. In *Proc. IEEE Aerospace Conference*, Big Sky, Montana (USA), 9–16 March 2003.
33. G. Deen, T. Lehman, and J. Kaufman. The almaden optimal grid project. In *IEEE Autonomic Computing Workshop (5th AMS)*, Seattle, Washington (USA), 14–21 June 2003.
34. M. Dorigo and L. M. Gambardella. Ant colonies for the traveling salesman problem. *BioSystems*, 43:73–81, 1997.
35. M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, Massachusetts (USA), 2004.
36. R. J. Doyle. Attention focusing and anomaly detection in systems monitoring. In *Proc. Third International Symposium on Articial Intelligence, Robotics, and Automation for Space (I-SAIRAS 94)*, pages 57–60, Pasadena, California (USA), 18-20 April 1994. NASA Jet Propulsion Laboratory.
37. A. S. Driesman, B. S. Ballard, D. E. Rodriguez, and S. J. Offenbacher. STEREO observatory trade studies and resulting architecture. In *Proc. IEEE Aerospace Conference*, volume 1, pages 63–80, Big Sky, Montana (USA), March 2001.
38. O. Etzioni. A Softbot-based interface to the Internet. *Communications of the ACM*, pages 72–76, July 1994.
39. R. J. Firby. The RAP language manual. Animate Agent Project Working Note AAP-6. Technical report, University of Chicago, Chicago, IL (USA), 1995.
40. D. A. Fullford. Distributed interactive simulation: its past, present, and future. In *Proc. WSC '96: 28th Conference on Winter Simulation*, pages 179–185, Washington, DC (USA), 1996. IEEE Computer Society.

41. A. G. Ganek. Autonomic computing: implementing the vision. Keynote presentation, Autonomic Computing Workshop, AMS 2003, 25 June 2003.
42. A. G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1):5–18, 2003.
43. E. Gat. ESL: A language for supporting robust plan execution in embedded autonomous agents. In *Proc. 11th International Conference on Parallel and Distributed Systems (ICPADS-2005), Workshop on Reliability and Autonimic Management in Parallel and Distributed Systems (RAMPDS-05), AAAI Fall Symposium on Plan Execution*, 1996.
44. M. R. Genesereth. Knowledge Interchange Format. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference*, pages 599–600, Cambridge, Massachusetts (USA), 1991. Morgan Kaufmann.
45. M. R. Genesereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48ff, July 1994.
46. J. C. Giarratano and G. D. Riley. *Expert Systems: Principles and Programming*. PWS Publishing Company, Boston, Massachusettes (USA), 4th edition, 15 October 2004.
47. M. D. Griffin and J. R. French, editors. *Reducing Space Mission Costs*. American Institute of Aeronautics and Astronautics, 2004.
48. D. Ground and J. Schwab. Concept evaluation program of simulation networking (SIMNET). Technical Report Report 86-CEP345, Army and Engineering Board, Fort Knox, Kentucky (USA), 1988.
49. R. Guitierrez and M. Huhns. Achieving software robustness via multiagent-based redundancy. In *IJCAI Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems*, Acapulco (Mexico), 10 August 2003.
50. H. Guo. A Bayesian approach for autonomic algorithm selection. In *IJCAI Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems*, Acapulco (Mexico), 10 August 2003.
51. J. J. Guzman and A. Edery. Mission design for the MMS tetrahedron formation. In *Proc. IEEE Aerospace Conference*, volume 1, pages 540–545, Big Sky, Montana (USA), March 2004.
52. D. Harel. Comments made during presentation at "Formal Approaches to Complex Software Systems" panel session. In *ISoLA-04 First International Conference on Leveraging Applications of Formal Methods*, Paphos (Cyprus). 31 October 2004.
53. L. Herger, K. Iwano, P. Pattnaik, J. J. Ritsko, and A. G. Davis. Special issue (John J. Ritsko, Editor-in-Chief): Autonomic computing. *IBM Systems Journal*, 42(1), 2003.
54. Hewlett-Packard Development Company, L.P. Adaptive infrastructure. *HP World*, 11–15 August 2003.
55. D. E. Hiebeler. The swarm simulation system and individual-based modeling. In *Proc. Decision Support 2001: Advanced Technology for Natural Resource Management*, Toronto (Canada), September 1994.
56. M. Hinchey, J. Rash, and C. Rouff. Verification and validation of autonomous systems. In *Proc. SEW-26, 26th Annual NASA/IEEE Software Engineering Workshop*, pages 136–144, Greenbelt, Maryland (USA), November 2001. NASA Goddard Space Flight Center, Greenbelt, Maryland (USA), IEEE Computer Society Press, Los Alamitos, California (USA).

57. M. G. Hinchey, J. L. Rash, and C. A. Rouff. Enabling requirements-based programming for highly dependable complex parallel and distributed systems. In *Proc. 1st International Workshop on Distributed, Parallel and Network Applications (DPNA 2005)*, Fukuoka (Japan), 20–22 July 2005. IEEE Computer Society Press, Los Alamitos, California (USA).
58. M. G. Hinchey, J. L. Rash, and C. A. Rouff. A formal approach to requirements-based programming. In *Proc. IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2005)*. IEEE Computer Society Press, Los Alamitos, California (USA), 3–8 April 2005.
59. M. G. Hinchey, J. L. Rash, and C. A. Rouff. Towards an automated development methodology for dependable systems with application to sensor networks. In S. F. Andler and A. Cervin, editors, *Proc. Real Time in Sweden 2005 (RTiS2005), the 8th Biennial SNART Conference on Real-time Systems (Reprinted from Proc. IEEE Workshop on Information Assurance in Wireless Sensor Networks (WSNIA 2005), Proc. International Performance Computing and Communications Conference (IPCCC-05), 2005)*, Skövde University Studies in Informatics, pages 73–79, University of Skövde (Sweden), 2005.
60. T. Hoare and R. Milner. Grand challenges for computing research. *Comput. J.*, 48(1):49–52, 2005.
61. W. M. L. Holcombe. Mathematical models of cell biochemistry. Technical Report CS-86-4, Sheffield University, UK, 1986.
62. W. M. L. Holcombe. Towards a formal description of intracellular biochemical organization. Technical Report CS-86-1, Sheffield University, UK, 1986.
63. P. Horn. Autonomic computing: IBM's perspective on the state of information technology. White paper, IBM Research, Armonk, New York (USA), October 2001.
64. R. S. Hornstein, J. K. Willoughby, J. A. Gardner, R. Casasanta, J. Donald J. Hei, F. J. Hawkins, J. Eugene S. Burke, J. E. Todd, J. A. Bell, and R. E. Miller. Cost efficient operations: Challenges from NASA administrator and lessons learned from "hunting sacred cows". In *Fourth International Symposium on Space Mission Operations and Ground Data Systems*, Munich (Germany), 16–20 September 1996. American Aeronautical Society.
65. P. Hughes, G. Shirah, and E. Luczak. Advancing satellite operations with intelligent graphical monitoring systems. In *Proc. AIAA Computing in Aerospace Conference*, San Diego, California (USA), October 1993.
66. P. M. Hughes. Application of autonomic computing concepts for GSFC's next generation missions. Presentation at the Woodrow Wilson International Center for Scholars, 28 October 2003.
67. M. N. Huns, V. T. Holderfield, and R. L. Z. Gutierrez. Robust software via agent-based redundancy. In *Proc. Second International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2003)*, pages 1018–1019, Melbourne, Victoria (Australia), 14–18 July 2003.
68. IBM. Autonomic computing concepts. White paper, 2001.
69. IBM. An architectural blueprint for autonomic computing. White paper, April, revised October, 2003.
70. IBM. Special issue on autonomic computing. *IBM Systems Journal*, 42(1), 2003.
71. IBM and Cisco Systems. Adaptive Services Framework. White paper, version 1.0, IBM and Cisco Systems, 14 October 2003.

72. *IEEE International Conference on Autonomic Computing (ICAC'04)*, New York, New York USA, 17–18 May 2004.
73. T. Iida, J. N. Pelton, and E. Ashford. *Satellite Communications in the 21st Century: Trends and Technologies*. AIAA, 2003.
74. J. C. Isaacs. Next Generation Space Telescope mission: Operations Concept Document (OCD). Technical Report STScI-NGST-OPS-0001C, Space Telescope Science Institute, Baltimore, Maryland (USA), 2001.
75. N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, March 2000.
76. e. a. Joseph P. Pickett, editor. *American Heritage Dictionary of the English Language*. Houghton Mifflin, 4th edition, 2005.
77. G. Kaiser, J. Parekh, P. Gross, and G. Valetto. Kinesthetics extreme: An external infrastucture for monitoring distributed legacy systems. In *The Autonomic Computing Workshop, 5th Int. Workshop on Active Middleware Services (AMS'03)*, pages 22–30, Seattle, Washington (USA), 25 June 2003.
78. G. Karjoth. Access control with IBM Tivoli access manager. *ACM Trans. Inf. Syst. Secur.*, 6(2):232–257, 2003.
79. J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pages 1942–1948, TPiscataway, New Jersey (USA), 1995.
80. J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, January 2003.
81. K. Kistler-Glendon. Beginning the autonomic journey - a review and lessons learned from an autonomic computing readiness assessment at a major US telecom. In *Proc. CHIACS2: Conference on the Human Impact and Application of Autonomic Computing Systems*, Yorktown Heights, New York, 21 April 2004.
82. Y. Labrou and T. Finin. A proposal for a new KQML specification. Technical Report TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, Maryland (USA), 1997.
83. D. B. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.
84. G. Langranchi, P. D. Peruta, A. Perrone, and D. Calvanese. Toward a new lanscape of systems management in an autonomic computing environment. *IBM Systems Journal*, 42(1):38–44, 2003.
85. W. J. Larson and J. R. Wertz, editors. *Space Mission Analysis and Design*. Microcosm, Inc., and Kluwer Academic Publishers, 1992.
86. T. Lau, D. Oblinger, L. Bergman, V. Castelli, and C. Anderson. Learning procedures for autnomic computing. In *IJCAI Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems*, Acapulco (Mexico), 10 August 2003.
87. M. Lauriente, R. Durand, A. Vampola, H. C. Koons, and D. Gorney. An expert system for diagnosing anomalies of spacecraft. In *Proceedings of the Third International Symposium on Articial Intelligence, Robotics, and Automation for Space (I-SAIRAS 94)*, pages 63–66, Pasadena, California (USA), 18–20 April 1994. NASA Jet Propulsion Laboratory.
88. S. M. Lewandowski, D. J. V. Hook, G. C. O'Leary, J. W. Haines, and L. M. Rossey. SARA: Survivable autonomic response architecture. In *Proc. DARPA Information Survivability Conference and Exposition II, Volume 1*, pages 77–88, June 2001.

89. S. Lightstone. Towards benchmarking - autonomic computing maturity. In *Proc. IEEE Workshop on Autonomic Computing Principles and Architectures (AUCOPA 2003)*, pages 451–456, Banff, Alberta (Canada), 22–23 August 2003.
90. M. Littman, T. Nguyen, and J. Hirsh. A model of cost-sensitive fault mediation. In *IJCAI Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems*, Acapulco (Mexico), 10 August 2003.
91. H. Liu and M. Parashar. Dios++: A framework for rule-based autonomic management of distributed scientific applications. In *9th International Euro-Par Conference (Euro-Par 2003)*, volume 2790 of *Lectgure Notes in Computer Science*, pages 66–73. Springer-Verlag, 2003.
92. J. Llinas, C. Bowman, G. Rogova, A. Steinberg, E. Waltz, and F. White. Revisiting the JDL Data Fusion Model II. In *Proc. 7th International Data Fusion Conference*, Stockholm (Sweden), 28 June – 1 July 2004.
93. A. C. Long, J. O. Cappellari, Jr., C. E. Velez, and A. J. Fuchs, editors. *Goddard Trajectory Determination System (GTDS) Mathematical Theory, Revision 1.* NASA Goddard Space Flight Center, Greenbelt, Maryland (USA), 1989.
94. F. Luna and B. Stefansson. *Economic Simulations in Swarm: Agent-Based Modelling and Object Oriented Programming.* Kluwer Academic Publishers, 2000.
95. L. Lymberopoulos, E. Lupu, and M. Sloman. An adaptive policy-based framework for network services management. *Journal of Network and Systems Management*, 11(3), 2003.
96. Microsoft Corporation. Dynamic Systems Initiative overview. White paper, 31 March, revised 15 November, 2004.
97. M. T. Morrow, C. A. Woolsey, and G. M. Hagerman, Jr. Exploring Titan with autonomous, buoyancy driven gliders. *Journal of the British Interplanetary Society*, 59(1):27–34, January 2006.
98. R. Muralidhar and M. Parashar. A distributed object infrastructure for interaction and steering. *Concurrency and Computation: Practice And Experience*, 15(10):957–977, 2003.
99. R. Murch. *Autonomic Computing.* IBM Press, 2004.
100. N. Muscettola, P. P. Nayak, B. Pell, and B. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1/2):5–48, 1998.
101. P. P. Nayak, D. E. Bernard, G. Dorais, E. B. Gamble Jr., B. Kanefsky, J. Kurien, W. Millar, N. Muscettola, K. Rajan, N. Rouquette, B. D. Smith, W. Taylor, and Y. wen Tung. Validating the DS1 remote agent experiment. In *Proc. of the 5th International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS-99)*, 1999.
102. D. A. Norman, A. Ortony, and D. M. Russell. Affect and machine design: Lessons for the development of autonomous machines. *IBM Systems Journal*, 42(1):38–44, 2003.
103. J. Padget. The role of norms in autonomic organizations. In *IJCAI Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems*, Acapulco (Mexico), 10 August 2003.
104. M. Papazoglou, S. Laufmann, and T. K. Sellis. An organizational framework for cooperating intelligent information systems. *International Journal of Intelligent and Collaborative Information Systems*, 1(1):169–202, 1992.

105. M. Parashar, editor. *The Autonomic Computing Workshop, 5th Int. Workshop on Active Middleware Services (AMS 2003)*, Seattle, Washington (USA), 25 June 2003. Proceedings IEEE Computer Society.
106. D. Parkes. Five AI challenges in strategy proof computing. In *IJCAI Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems*, Acapulco (Mexico), 10 August 2003.
107. R. Patil, R. Fikes, P. Patel-Schneider, D. McKay, T. Finin, T. Gruber, and R. Neches. The DARPA Knowledge Sharing Effort: Progress report. In *Proc. KR'92, The Annual International Conference on Knowledge Acquisition*, pages 599–600, Cambridge, Massachusetts (USA), 1992.
108. L. D. Paulson. Computer system, heal thyself. *IEEE Computer*, 35(8):20–22, August 2002.
109. B. Pell, D. E. Bernard, S. A. Chien, E. Gat, N. Muscettola, P. P. Nayak, M. D. Wagner, and B. C. Williams. An autonomous spacecraft agent prototype. *Autonomous Robots*, 5(1):29–52, March 1998.
110. J. Pitt and A. Mamdani. Some remarks on the semantics of FIPA's Agent Communication Language. *Autonomous Agents and Multi-Agent Systems*, 2(4):333–356, 1999.
111. J. L. Rash, M. G. Hinchey, C. A. Rouff, D. Gračanin, and J. D. Erickson. Experiences with a requirements-based programming approach to the development of a NASA autonomous ground control system. In *Proc. IEEE Workshop on Engineering of Autonomic Systems (EASe 2005) held at the IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2005)*. IEEE Computer Society Press, Los Alamitos, California (USA), 3–8 April 2005.
112. J. L. Rash, M. G. Hinchey, C. A. Rouff, and D. Gračanin. Formal requirements-based programming for complex systems. In *Proc. International Conference on Engineering of Complex Computer Systems*, Shanghai (China), 16–20 June 2005. IEEE Computer Society Press, Los Alamitos, California (USA).
113. J. L. Rash, C. A. Rouff, W. F. Truszkowski, D. Gordon, and M. G. Hinchey, editors. *Formal Approaches to Agent-Based Systems, First International Workshop, FAABS 2000, Greenbelt, Maryland (USA), April, 2000, Revised Papers*, volume 1871 of *Lecture Notes in Computer Science, Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin (Germany) and Heidelburg (Germany) and New York, New York (USA), 2001.
114. M. D. Rayman, P. Varghese, D. H. Lehman, and L. L. Livesay. Results from the Deep Space 1 technology validation mission. *Acta Astronautica*, 47(2–9), July 2000.
115. C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.
116. C. K. Riesbeck and R. C. Schank. *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Inc., 1989.
117. C. Rouff. A test agent for testing agents and their communities. In *Proc. IEEE Aerospace Conference*, Big Sky, Montana (USA), March 2002.
118. C. Rouff and M. Hinchey. Modeling the LOGOS multi-agent system with CSP. In *Proc. AAAI Spring Symposium, Technical Report SS-01-04*, 2001.
119. C. Rouff, J. Rash, M. Hinchey, and W. Truszkowski. Formal methods at NASA Goddard Space Flight Center. In *Agent Technology from a Formal Perspective*, NASA Monographs in Systems and Software Engineering, pages 287–310. Springer-Verlag, London (UK), 2005.

120. C. Rouff and W. Truszkowski. A process for introducing agent technology into space missions. In *Proc. IEEE Aerospace Conference*, March 2001.
121. C. Rouff, A. Vanderbilt, M. Hinchey, W. Truszkowski, and J. Rash. Properties of a formal method for prediction of emergent behaviors in swarm-based systems. In *Proc. 2nd IEEE International Conference on Software Engineering and Formal Methods*, Beijing (China), September 2004.
122. C. A. Rouff. Autonomy in future space missions. In *Proc. IEEE Aerospace Conference*, Big Sky, Montana (USA), March 2003.
123. C. A. Rouff, M. G. Hinchey, J. L. Rash, W. F. Truszkowski, and D. Gordon-Spears, editors. *Agent Technology from a Formal Perspective*. NASA Monographs in Systems and Software Engineering. Springer-Verlag, London (UK), 2006.
124. C. A. Rouff, J. L. Rash, and M. G. Hinchey. Experience using formal methods for specifying a multi-agent system. In *Proc. Sixth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2000)*, Tokyo (Japan), 2000. IEEE Computer Society Press, Los Alamitos, California (USA).
125. C. A. Rouff, W. F. Truszkowski, M. G. Hinchey, and J. L. Rash. Verification of NASA emergent systems. In *Proc. 9th IEEE International Conference on Engineering of Complex Computer Systems*, Florence (Italy), April 2004. IEEE Computer Society Press, Los Alamitos, California (USA).
126. C. A. Rouff, W. F. Truszkowski, J. L. Rash, and M. G. Hinchey. A survey of formal methods for intelligent swarms. Technical Report TM-2005-212779, NASA Goddard Space Flight Center, Greenbelt, Maryland (USA), 2005.
127. L. Russell, S. Morgan, and E. Chron. Clockwork: A new movement in autonomic systems. *IBM Systems Journal*, 42(1):77–84, 2003.
128. L. Russell, S. Morgan, and E. Chron. On-line model selection procedures in Clockwork. In *Proc. IJCAI Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems*, Acapulco (Mexico), 10 August 2003.
129. R. SAhoo, I. Rish, A. Oliner, M. Gupta, J. Moreira, S. Ma, R. Vilata, and A. Sivasubramaniam. Autonomic computing features for large-scale server management and control. In *IJCAI Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems*, Acapulco (Mexico), 10 August 2003.
130. M. Salehie and L. Tahvildari. Autonomic computing: emerging trends and open problems. *SIGSOFT Softw. Eng. Notes*, 30(4):1–7, 2005.
131. M. Savage and M. Askenazi. Arborscapes: A swarm-based multi-agent ecological disturbance model. Working paper 98-06-056, Santa Fe Institute, Santa Fe, New Mexico (USA), 1998.
132. P. Scerri, D. Pynadath, and M. Tambe. Towards adjustable autonomy for the real-world. *Journal of AI Research*, vol. 17, 2002.
133. T. P. Schetter, M. E. Campbell, and D. M. Surka. Multiple agent-based autonomy for satellite constellations. *Artifitial Intellegence*, 145(1-2):147–180, 2003.
134. U. M. Schwuttke, J. R. Veregge, and A. G. Quan. Performance results of cooperating expert systems in a distributed real-time monitoring system. In *Proc. Third International Symposium on Articial Intelligence, Robotics, and Automation for Space (I-SAIRAS 94)*, pages 79–83, Pasadena, California (USA), 18–20 April 1994. NASA Jet Propulsion Laboratory.

135. R. Sherwood, J. Wyatt, H. Hotz, A. Schlutsmeyer, and M. Sue. Lessons learned during implementation and early operations of the ds1 beacon monitor experiment. In *Proc. Third International Symposium on Reducing the Cost of Ground Systems and Spacecraft Operations*, Tainan (Taiwan), 1999.
136. H. A. Simon. *Models of Thought*, chapter on "Motivational and emotional controls of cognition", pages 29–38. Yale University Press, 1979 (reprinted).
137. R. F. Simpson. Explorer platform. In *AIAA Aerospace Sciences Meeting*, Reno, Nevada (USA), 11-14 January 1988. AIAA.
138. R. L. Simpson. A computer model of case-based reasoning in problem solving: An investigation in the domain of dispute mediation. Technical Report GIT-ICS-85/18, Georgia Institute of Technology, School of Information and Computer Science, Atlanta GA, 1985.
139. A. Sloman. Review of: Rosalind Picard's Affective Computing, MIT Press, 1997. *AI Magazine*, pages 127–137, Spring 1999.
140. A. Sloman and M. Croucher. Why robots will have emotions. In *Proc. 7th International Joint Conference on Artificial Intelligence*, pages 197–202, Vancouver (Canada), 1981.
141. M. Smirnov. Area: Autonomic communications. In *Proc. Consultation meeting on Future Emerging Technologies: "New Communication Paradigms for 2020"*, Brussels (Belgium), March 2004. European Union.
142. M. Smirnov and R. Popescu-Zeletin. Autonomic communication. In *Proc. New Communication Paradigms for 2020*, "Brainstorming" meeting on Future Emerging Technologies, Brussels (Belgium), July 2003. European Union.
143. D. Smith. Efficient mission control for the 48-satellite Globalstar constellation. In *Proc. Third International Symposium on Space Mission Operations and Ground Data Systems*, pages 663–670, Greenbelt, Maryland (USA), 15–18 November 1994.
144. J. B. Smith. *Collective Intelligence in Computer-Based Collaboration*. CRC, 1994.
145. R. Sterrit and M. G. Hinchey. SPAACE:: Self-properties for an autonomous & autonomic computing environment. In *Proc. The 2005 International Conference on Software Engineering Research and Practice (SERP'05)*, pages 9–15, Las Vegas, Nevada (USA), 27 June 2005. CSREA Press.
146. R. Sterritt. Towards autonomic computing: Effective event management. In *Proc. 27th Annual IEEE/NASA Software Engineering Workshop (SEW)*, pages 40–47, Greenbelt, Maryland (USA), 3–5 December 2002. IEEE Computer Society Press, Los Alamitos, California (USA).
147. R. Sterritt. Autonomic computing: the natural fusion of soft computing and hard computing. In *IEEE International Conference on Systems, Man and Cypernetics*, pages 4754–4759, Washington, D.C. (USA), 5–8 October 2003.
148. R. Sterritt. Pulse monitoring: Extending the health-check for the autonomic GRID. In *IEEE Workshop on Autonomic Computing Principles and Architectures (AUCOPA 2003)*, pages 433–440, Banff, Alberta (Canada), 22–23 August 2003.
149. R. Sterritt. xACT: Autonomic computing and telecommunications. BT Exact Research Fellowship report, British Telecommunications, 2003.
150. R. Sterritt. Autonomic networks: Engineering the self-healing property. *Engineering Applications of Artificial Intelligence*, 17(7):727–739, 2004.

151. R. Sterritt. SelfWares-Episode IV—Autonomic computing: A new hope. Presentation, IS&T Colloquium, NASA Goddard Space Flight Center, Greenbelt, Maryland (USA), 1 December 2004.
152. R. Sterritt and D. F. Bantz. PAC-MEN: Personal autonomic computing monitoring environments. In *Proc. IEEE DEXA 2004 Workshops – 2nd International Workshop on Self-Adaptive and Autonomic Computing Systems (SAACS04)*, pages 737–741, Zaragoza (Spain), 30 August – 03 September 2004. IEEE.
153. R. Sterritt and D. Bustard. Fusing hard and soft computing for fault mangement in telecommunications systems. *IEEE Transactions on Systems Man and Cybernetics – Part C: Applications and Reviews*, 32(2):92–98, May 2002.
154. R. Sterritt and D. Bustard. Towards an autonomic computing environment. In *1st International Workshop on Autonomic Computing Systems*, pages 694–698, Prague (Czech Republic), 1–5 September 2003.
155. R. Sterritt, D. Bustard, and A. McCrea. Aunonomic computing correlation for fault management system evolution. In *Proc. IEEE International Conference on Industrial Informatics (INDIN 2003)*, pages 240–247, Banff, Alberta (Canada), 21-24 August 2003.
156. R. Sterritt and D. W. Bustard. Autonomic computing—a means of achieving dependability? In *Proc. IEEE International Conference on the Engineering of Computer Based Systems (ECBS-03)*, pages 247–251, Huntsville, Alabama (USA), April 2003. IEEE Computer Society Press, Los Alamitos, California (USA).
157. R. Sterritt and S. Chung. Personal autonomic computing self-healing tool. In *Proc. IEEE Workshop on the Engineering of Autonomic Systems (EASe 2004) at the 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2004)*, pages 513–520, Brno (Czech Republic), May 2004.
158. R. Sterritt, G. Garrity, E. Hanna, and P. O'Hagan. Survivable security systems through autonomicity. In *Proc. Workshop on Radical Agent Concepts (WRAC) 2005*, number 3825 in Lecture Notes in Computer Science, Greenbelt, Maryland (USA), September 2005. Springer-Verlag.
159. R. Sterritt, D. Gunning, A. Meban, and P. Henning. Exploring autonomic options in an unified fault management architecture through reflex reactions via pulse monitoring. In *Proc. IEEE Workshop on the Engineering of Autonomic Systems (EASe 2004) at the 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2004)*, pages 449–455, Brno (Czech Republic), May 2004.
160. R. Sterritt and M. Hinchey. Engineering ultimate self-protection in autonomic agents for space exploration missions. In *Proc. EASe-2005, 2nd IEEE Workshop on Engineering Autonomic Systems, at ECBS 2005, 12th IEEE International Conference on Engineering of Computer Based Systems*, pages 506–511, Greenbelt, Maryland (USA), 4–7 April 2005. IEEE Computer Society Press, Los Alamitos, California (USA).
161. R. Sterritt and M. G. Hinchey. Apoptosis and self-destruct: A contribution to autonomic agents? In *Proc. FAABS-III, 3rd NASA/IEEE Workshop on Formal Approaches to Agent-Based Systems*, pages 269–278. Springer-Verlag, April 2004.
162. R. Sterritt and M. G. Hinchey. Birds of a feather session: "autonomic computing: Panacea or poppycock?". In *Proc. EASe 2005: IEEE Workshop on the*

Engineering of Autonomic Systems at 12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2005), Greenbelt, Maryland (USA), 3–8 April 2005.
163. D. J. T. Sumpter, G. B. Blanchard, and D. S. Broomhead. Ants and agents: a process algebra approach to modelling ant colony behaviour. *Bulletin of Mathematical Biology*, 63(5):951–980, September 2001.
164. Sun Microsystems. N1 - introducing just-in-time computing. White paper, 2002.
165. M. Swartwout. Engineering data summaries for space missions.
166. S. J. Talabac. Spacecraft constellations: The technological challenges in the new millennium. Technical report, AETD Information Systems Center, Code 588, NASA Goddard Space Flight Center, Greenbelt, Maryland (USA), 27 September 1999.
167. H. D. Tananbaum, N. E. White, J. A. Bookbinder, F. E. Marshall, and F. Cordova. Constellation X-ray mission: implementation concept and science overview. In O. H. Siegmund and K. A. Flanagan, editors, *Proc. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 3765 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 62–72, October 1999.
168. M. Therani, D. Zeng, and M. Dror. Decentralized resource management in autonomic systems. In *IJCAI Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems*, Acapulco (Mexico), 10 August 2003.
169. H. Tianfield. Multi-agent based autonomic architecture for network management. In *Proc. IEEE International Conference on Industrial Informatics*, pages 462–469, August 2003.
170. H. Tianfield and R. Unland (guest eds.). Special issue: Autonomic computing systems. *Engineering Applications of Artificial Intelligence*, 17(7):689–869, 2004.
171. R. L. Ticker and D. McLennan. NASA's New Millennium Space Technology 5 (ST5) project. In *IEEE Aerospace Conference*, volume 7, pages 609–617, Big Sky, Montana (USA), March 2000.
172. C. N. Toomey, E. Simoudis, R. W. Johnson, and W. S. Mark. Software agents for the dissemination of remote terrestrial sensing data. In *Proc. Third International Symposium on Articial Intelligence, Robotics, and Automation for Space (I-SAIRAS 94)*, pages 19–22, Pasadena, California (USA), 18–20 April 1994. NASA Jet Propulsion Laboratory.
173. W. Trumler, F. Bagci, J. Petzold, and T. Ungerer. Smart doorplates - toward an autonomic computing system. In *The Autonomic Computing Workshop, 5th Int. Workshop on Active Middleware Services (AMS'03)*, Seattle, Washington (USA), 25 June 2003. IEEE Computer Society.
174. W. Trumler, J. Petzold, F. Bagci, and T. Ungerer. AMUN - autonomic middleware for ubiquitous environments applied to the Smart Doorplate project. In *International Conference on Autonomic Computing (ICAC'04)*, pages 274–275, May 2004.
175. W. Truszkowski and H. Hallock. Agent technology from a NASA perspective. In *Proc. CIA-99, Third International Workshop on Cooperative Information Agents*, Uppsala (Sweden), July, August 1999. Springer-Verlag.

176. W. Truszkowski, M. Hinchey, J. Rash, and C. Rouff. NASA's swarm missions: The challenge of building autonomous software. *IEEE IT Professional*, 6(5):47–52, September/October 2004.
177. W. Truszkowski and C. Rouff. An overview of the NASA LOGOS and ACT agent communities. In *Proc. World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, Florida (USA), July 2001.
178. W. Truszkowski and C. Rouff. Progressive autonomy. In *Proc. 2002 NASA/IEEE Software Engineering Workshop*, Greenbelt, Maryland (USA), December 2002. IEEE Press.
179. W. Truszkowski, C. Rouff, S. Bailin, and M. Riley. Progressive autonomy: A method for gradually introducing autonomy into space missions. *Innovations in Systems and Software Engineering*, 1(2):89–99, September 2005.
180. W. F. Truszkowski, L. Hallock, C. A. Rouff, J. Kerlin, J. L. Rash, M. G. Hinchey, and R. Sterritt. *Autonomous and Autonomic Systems with Applications to NASA Intelligent Spacecraft Operations and Exploration Systems*. NASA Monographs in Systems and Software Engineering. Springer-Verlag, London (UK), 2007.
181. W. F. Truszkowski, M. G. Hinchey, J. L. Rash, and C. A. Rouff. Autonomous and autonomic systems: A paradigm for future space exploration missions. *IEEE Transactions on Systems Man and Cybernetics – Part C: Applications and Reviews*, 36(3), May 2006.
182. W. F. Truszkowski, J. L. Rash, C. A. Rouff, and M. G. Hinchey. Asteroid exploration with autonomic systems. In *Proc. 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS), Workshop on Engineering of Autonomic Systems (EASe)*, Brno (Czech Republic), May 2004. IEEE Computer Society Press, Los Alamitos, California (USA).
183. W. F. Truszkowski, J. L. Rash, C. A. Rouff, and M. G. Hinchey. Some autonomic properties of two legacy multi-agent systems — LOGOS and ACT. In *Proc. 11th IEEE International Conference on Engineering Computer-Based Systems (ECBS), Workshop on Engineering Autonomic Systems (EASe)*, pages 490–498, Brno (Czech Republic), May 2004. IEEE Computer Society Press, Los Alamitos, California (USA).
184. W. F. Truszkowski, C. A. Rouff, S. Bailin, and M. Rilee. Progressive autonomy—An incremental agent-based approach. In *Proc. 2005 International Conference on Software Engineering Research and Practice (SERP'05)*, pages 9–15, Las Vegas, Nevada (USA), 27 June 2005. CSREA Press.
185. K. Tumer, A. Agogino, and D. Wolpert. *Autonomous Agents & Multiagent Systems, Part 1*, chapter "Learning sequences of actions in collectives of autonomous agents", pages 378–385. ACM press, 2002.
186. C. Vanek. TDRSS, space-based communications for the present and for the future. In *Proc. Space Programs and Technologies Conference and Exhibit*, Huntsville, Alabama (USA), 21–23 September 1993.
187. R. Want, T. Pering, and D. Tennenhouse. Comparing autonomic and proactive computing. *IBM Systems Journal*, 42(1):129–135, 2003.
188. M. Weiser. Creating the invisible interface. In *Proc. UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology*, invited talk, page 1, Marina del Rey, California (USA), 1994. ACM Press.
189. J. R. Wertz, editor. *Spacecraft Attitude Determination and Control*. Kluwer Academic Publishers, 1978.

190. J. R. Wertz, J. L. Cloots, J. T. Collins, S. D. Dawson, G. Gurevich, B. K. Sato, and J. Hansen. Autonomous orbit control: Initial flight results from UoSAT-12. In *23rd Annual AAS Guidance and Control Conference, AAS 00-011*, Breckenridge, Colorado (USA), February 2-6 2000. American Aeronautical Society.
191. J. R. Wertz and W. J. Larson, editors. *Reducing Space Mission Costs*. Microcosm, Inc., and Kluwer Academic Publishers, 1996.
192. F. White. A model for data fusion. In *1st National Symposium on Sensor Fusion*, volume 2, 5–8 April 1988.
193. B. C. Williams and P. P. Nayak. Immobile robots: Artificial intelligence in the new Millennium. *AI Magazine*, 17(3):16–35, 1996.
194. B. C. Williams and P. P. Nayak. A model-based approach to reactive self-configuring systems. In *Procs. AAAI-96*, pages 971–978, 1996.
195. J. Wyatt, R. Sherwood, M. Sue, and J. Szijjarto. Flight validation of on-demand operations: The Deep Space One beacon monitor operations experiment. In *5th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS '99)*, ESTEC, Noordwijk (The Netherlands), 1–3 June 1999.

Index

Ada, 59
adaptive
  algorithms, 176
  operations, 185
  reasoning, 70
  scheduler, 135–137
  user interface, 69
Advanced Research Projects Agency (ARPA), 171
agent, 17–19, 22, 35, 97, 108, 147, 212
  actions, 107, 108, 149, 150
  adaptation, 186, 203
  aglets, 172, 173
  applets, 172
  architecture, 69–72, 75, 79, 81–83, 173
  assurance, 208
  attributes, 17, 18
  autonomy, 91
  based
    control centers, 69
    programming, 172
    spacecraft, 115
  behavior, 82
  beliefs, 149
  cloning, 72, 172, 205, 208, 209
  collaboration, 76, 103
  communication, 70, 75, 76, 83, 86, 108, 109, 173, 206
  communication language, 69, 72, 76–78, 83, 84, 103, 171, 173
  communities, 78, 79, 89, 197, 205, 206, 209, 234, 252
constellation, 201, 252
cooperation, 79, 241
development, 138, 206
effector, 12, 22, 82, 85, 86, 88, 179, 180, 184
efficiency, 203
executive, 119, 134, 243–245, 247, 248, 253
external interface, 78
fault tolerance, 208
formal verification, 91
Foundations of Intelligent Physical Agents (FIPA), 86, 173
framework, 87
goals, 71, 84, 149, 153, 154, 169
immobots, 17–19, 23, 107, 145, 146
incubator, 206
informational, 19, 20, 146
learning, 83, 151
migration, 72, 91, 198, 205, 206, 208, 209
mobile, 19, 172, 191
model, 70, 71
  domain, 83
  self, 151
  world, 149, 150
monitoring, 202, 229
multi-, 69, 72, 74, 75, 91
  satellite organization, 203
negotiation, 21, 128, 167, 168, 202, 252, 253
parallelization, 208
perception, 108, 110, 150

perceptor, 82, 83, 85–88, 179
persistence, 72
personal assistant, 20, 21
policy, 75
proxy, 89–91, 194, 197, 198, 208
reactive, 81, 82
reasoning, 83
remote, 35, 62, 115–119, 121–128, 130–132, 134–144, 169, 170, 241, 248, 249, 253
event-driven, 137
requirements, 110
robots, 18, 19, 22, 23, 107
robust, 82
sensing, 109, 110
spacecraft operations, 73
testing, 206, 208
trust, 197, 209
verification and validation, 206
Agent Concept Testbed (ACT), 69, 81–83, 87–89, 91, 194, 198
Agent-based Flight Operations Associate (AFLOAT), 69–72, 74–76, 78, 80, 91
archiving, 25, 66
artificial intelligence, 95, 182
assembly language, 55
asteroid belt, 3, 4, 38, 130, 142, 144, 214, 217–219, 221, 228
automation, 3, 5, 6, 9, 11, 14, 26, 28, 31, 32, 66, 67, 69, 141, 158, 161–163, 166, 169, 176, 183, 193, 195, 228
definition, 10
autonomic, 4, 5, 9–14, 23, 48, 95, 175–185, 187, 188, 191, 197, 199–201, 211, 215, 220–225, 227–229, 232, 234
agents, 185
architecture, 185
Astrolabe, 184
AutoMate, 185
Cisco Adaptive Network Care, 186
communications, 186, 187
computing, 176, 179, 180, 182, 185, 187, 188
control loop, 179
definition, 11
dependability, 176
Distributed Interactive Object Substrate, 185
economics, 187
element, 179–181, 183, 186, 187
environment, 175–180, 182–186
heart-beat monitor, 180
HP Adaptive Infrastructure, 186
Information Assurance, 13
initative, 180
Intel Proactive Computing, 186
Kinesthetics eXtreme, 184
learning, 179, 184
legacy systems, 184, 185
machine design, 182
maintainability, 176
manager, 179, 180
Microsoft Dynamic Systems Initiative, 186
Monitor, Analyze, Plan and Execute (MAPE), 179
monitoring and control, 183, 184
Nervous System (ANS), 175
Personal Computing, 187
policies, 177, 178, 185
pulse monitor, 180
reliability, 176
safety, 176
security, 176
self-
adaptation, 221
adjusting, 12, 177–179, 221
awareness, 4, 11, 65, 144, 176–178, 180, 221
chop, 177, 188, 221
configuring, 11, 176–178, 183–185, 214, 221
directed, 11, 12, 220
direction, 11, 65
governing, 11–13, 175, 185, 199, 220
healing, 11, 177, 178, 181, 184, 185, 221
managing, 11, 13, 175–180, 184, 185, 187, 188, 193, 198
modifying, 225
monitoring, 12, 32, 177–179, 221
motivating, 83
optimizing, 11, 177, 178, 182, 183, 185, 188, 221
organizing, 186

properties, 221
protecting, 11, 177, 178, 185, 221
regulating, 11
selfishness, 187
situated, 11
training, 183
ware, 12, 13, 176, 179
x, 12, 177, 179, 180, 185, 187, 188
Situated and Autonomic Communications Program, 186
Smart Doorplates, 183
Sun N1, 186
survivability, 175, 176
swarm, 220
Tivoli management environment, 183
ubiquitous computing, 186
autonomy
adjustable, 17
All Sky Monitor, 60
constellation, 198
cooperative, 113
debugging, 229
definition, 9
development, 229
flight, 38
mixed, 17
pointing control, 43
semi-autonomous, 198
testing, 222, 228
validation, 229
verification, 222, 229

Bayesian
networks, 183, 184
reasoning, 104, 105
statistics, 105
black holes, 192, 230

C++, 59
C-Language Integrated Production System (CLIPS), 75, 77, 78
Campbell, Mark, 202
charge-coupled device, 22, 124
chi-squared, 122, 127
Clockwork, 183
collaboration, 69, 95, 128, 145, 147, 158, 170, 171, 188, 197, 211, 216, 238
languages, 103
planning, 159, 161, 165
collective intelligence, 202
command and data handling, 34, 48, 49, 118, 123, 126
command sequence
generation, 163, 166
planning, 13
commercial off the shelf, 28, 30, 32, 138
common information model, 183
Communicating Sequential Processes (CSP), 91
communications, 126, 128, 134, 140, 141, 145
beacon mode, 128, 133
compression, 243
ground to space, 248
management, 33, 34
protocols, 192
space to ground, 245, 246
uplink and downlink, 25
complex systems, 176, 177, 212, 215, 223
constellation, 14, 46, 63, 65, 81, 82, 89, 132, 138, 144, 149, 165, 175, 191–197, 199, 200, 202, 209, 230, 231, 234, 241, 248–253
advantages, 195
agents, 197, 201, 203, 252
autonomic, 199, 201
autonomy, 197, 198
clusters, 149, 196, 202
communications, 195, 201, 203, 231, 244, 246, 250, 252, 253
community, 197
complex, 191
control, 201, 202
coordination, 205
costs, 194–196
formation flying, 16, 17, 46, 63, 65, 137, 146, 149, 151, 155, 165, 191, 192, 195, 196, 200, 230
governance, 191, 200
ground stations, 197
learning, 200, 201
low earth orbit, 196
management, 234
microsat, 231
military applications, 196
missions
Cluster, 193

Constellation X, 191
ESCORT, 193
Experimental Spacecraft System (XSS), 193
Global Positioning System (GPS), 196
GlobalStar, 15, 193, 196
Iridium, 14, 15
Magnetospheric Multiscale, 193
Magnetotail Constellation, 194
Orbcomm, 193
Orbital Express, 193
ST5, 191
Total Internal Reflection Optical System, 193
organization, 203, 205
overview, 194
simple, 191
simulation, 197
status, 200
types, 193
Continuous Process Improvement, 250
control
systems, 104
theory, 182
cooperation, 70, 111, 114, 119, 124, 129–132, 139, 144–152, 156, 159, 160, 162–166, 168–173, 182, 197, 198, 228, 241, 248
actions, 155
hierarchical, 153–155, 159, 161
mission planning, 159
model, 149, 156, 166
perception, 156
planning, 152, 153
problem solving, 170, 171
science planning, 159
spacecraft, 166
technologies, 152, 169
virtual platform, 165, 167
coordination, 119, 151, 155, 161, 171, 193, 195, 202, 203, 228, 251
architecture, 203
cost
lifecycle, 53
savings, 15, 134, 135
crew, 3, 26, 37, 38, 115, 228, 232, 234, 241
data
analysis, 26, 35, 125, 148, 159, 163
archiving, 26, 66, 234
calibration, 26
capture, 25, 26, 29, 66
collection, 166
compression, 44
distribution, 25
errors, 243
fusion, 110, 156, 159, 163, 166
integration, 252
latency, 253
management, 29
mining, 195
monitoring, 26, 33, 35, 66, 119, 120, 122, 123, 127, 131, 132, 134, 139–143, 242
monitoring and trending, 143, 243, 245, 247, 250
processing, 25, 26, 35, 117, 119, 120, 124, 128–130, 133, 134, 140–144, 195, 234, 243–245, 247, 252
onboard, 65
science, 157, 159
reduction processing, 251
storage, 29, 33, 34, 44, 45, 66, 116, 119, 120, 124, 125, 134, 140, 141, 144, 244–246
trending, 119–123, 125, 127, 128, 131, 132, 134, 139–143
validation, 66, 245
verificatoin, 245
deadlock, 134
decision support, 234
Deep Space Network, 245, 246
Defense Advanced Research Projects Agency (DARPA), 13, 114, 170, 171
Autonomic Information Assurance, 13
Knowledge Sharing Effort, 103, 171
distributed computing, 182

e-commerce, 195
earth
centroid, 63
observation, 196
pointers, 28, 42, 45, 137
science plan, 232

Eberhart, Russell, 220
Einstein, Albert, 66, 230
Embedded Computing, 187
emergence, 205, 212, 213, 216, 220, 222–224
  definition, 212
  verification, 223
environment aware, 177, 178, 234
event correlation, 183
evolutionary computing, 179
expert system, 16, 35, 199
exploration, 38
  robotic, 3
Explorer Platform, 72

failure remediation, 184
fault
  correction, 26, 131–133
  detection, 11, 120, 127, 128, 131
  detection and correction, 26, 29, 32, 33, 35, 36, 47, 50, 55, 57–59, 61–63, 116, 118, 119, 122, 125, 126, 131, 134, 139, 141–143, 197, 242–247, 250
    constraints, 50
  diagnosis and correction, 26, 66
  tolerance, 11, 78, 176
Finin, Tim, 76
flight
  autonomy capabilities, 55
  computer, 28
  data storage, 130
  dynamics, 237
  dynamics team, 238
  hardware, 31, 32, 119, 135, 138
  operating system, 125
  operations team, 10, 25, 29, 37, 39, 51–53, 58, 61, 116, 135, 238, 246
  processor, 126
  software, 5, 10, 19, 29–33, 35–37, 40, 41, 43–45, 47–54, 57–64, 66, 67, 115–118, 120–127, 130, 134–139, 141–143, 191, 237–239, 241–245, 250, 253
    backbone, 115–125, 138, 139, 206, 243–245, 247, 250, 253
    bus, 230
    design, 249
    development, 137, 138, 141
    executive, 33, 34, 169
    internal data transfer, 44
    maintenance, 52, 137, 188
    operating systems, 59
    patch, 52
    reuse, 138
    safemode, 10, 21, 29, 32, 33, 35, 39, 45, 47, 50, 54, 55, 57, 59, 62, 116, 118–122, 131, 135, 143, 158, 161, 246
    testing, 137, 138
  system, 25, 26, 33, 34, 41–45, 55, 132
forecasting, 183
formal
  language, 103
  methods, 91, 223
  verification, 91
Formal Approaches to Swarm Technology (FAST), 223
formation flying, 165

gamma ray, 60, 62
General Theory of Relativity, 66, 230
genetic algorithms, 107, 108, 183, 221
GenSAA/Genie, 80, 81
geocentric inertial, 237
Georgia Tech, 169, 170, 173, 219
Global Positioning System (GPS), 33, 63, 120, 125, 126, 140, 196, 237, 246
goals, 85, 96, 98, 100, 154, 175
  directed, 70
  science, 158, 162, 165
gravitational radiation, 65
grid computing, 186
groupware, 163

health and safety, 10, 16, 25, 29, 33, 34, 39, 40, 46, 49, 50, 52, 53, 80, 88, 90, 115, 116, 118–120, 122–124, 138, 139, 143, 144, 158, 163, 191, 195, 200, 209, 247, 250, 253
health management, 145

IBM, 11, 172, 173, 176, 179–181, 183
  AlphaWorks Autonomic Zone, 183
image
  detector, 109
  multi-spectral, 109

processing, 109, 110
information systems, 234
intelligence, 182
ambient, 186
collective, 200
control, 171
distributed, 203
economical, 192, 199
emotion, 182
machine design, 12
interferometry, 64, 65, 231
laser, 230
UV-Optical Fizeau, 230
X-ray, 230
Internet, 172, 193, 220
invisible computing, 186, 187

Java, 173
Jet Propulsion Laboratory (JPL), 52, 61, 144, 169

Kalman filter, 56, 239
Kennedy, James, 220
knowledge
acquisition, 146
base, 70, 80, 81, 84, 171, 180, 202
capture, 183
Interchange Format (KIF), 171
management, 199
Query Manipulation Language (KQML), 76, 171, 173
representation, 183
sharing, 249
Sharing Effort, 103, 171

Lagrange point, 49, 64, 120, 136, 142, 217, 230, 232, 241
celestial pointer, 138, 142
Laufmann, Steve, 72, 76
learning, 72, 103, 106, 107, 183, 205, 249
adaptive, 169, 178, 179
case-based, 169
deliberative, 169
offline, 169
reactive, 169
similarity-based, 169
verification, 223
legacy systems, 184, 185, 188
Lights Out Ground Operations System (LOGOS), 69, 79–81, 91
logic
Boolean, 61
fuzzy, 104, 105, 137, 169, 221
tree, 42

magnetometer, 5, 61, 238
magnetosphere, 5, 7, 89, 194, 197
radio bursts, 231
Mars, 4, 18, 211
Mathematica, 105
middleware, 181, 184
mission
concept, 31, 211, 213, 214, 216, 218, 229, 234
control, 3–5, 10, 17, 25–27, 29, 31–35, 37–39, 41–43, 45–51, 53, 56, 58, 59, 63, 66, 67, 69, 72, 73, 81, 89–91, 115, 121, 123, 128–131, 133, 135–139, 141–145, 147, 160, 161, 164–166, 168, 191, 194, 195, 197, 200, 205–209, 218, 225, 227–229, 232, 234, 237, 238, 242–248, 250, 251
autonomy, 206
lights out, 5, 72, 115, 124, 139–141, 143, 244, 245, 250, 251
operations personnel, 29, 32
operator-to-spacecraft ratio, 15
costs, 15, 144, 146, 227, 238
design, 25, 201
future, 229, 232
heterogeneous, 15
homogeneous, 15
management, 156
manager, 164, 169
negotiation, 162
operations, 14, 69, 194, 227
risks, 146, 238
types
deep space, 144
robotic, 5
survey, 138, 142, 245
Mission Operations Control Center, 90
Mission Operations Planning and Scheduling System, 80, 81
missions

Application Lunar Base Activities, 214
Autonomous Nano Technology Swarm (ANTS), 3, 213, 214, 216–218, 221, 223–225
Big Bang Observer, 230
Black Hole Imager, 230
Cassini, 3
Cluster, 193
Compton Gamma Ray Observatory, 57, 58
Constellation X, 9, 191, 192, 230
Dawn, 4
deep space, 138
Deep Space One, 62, 144, 169
Earth Observing Spacecraft (EOS), 62, 63
Earth Observing-1, 33
Enceladus, 231
Extreme Ultraviolet Explorer, 58, 59, 78
Geospace Electrodynamic Connections, 8
Global Precipitation Measurement, 125, 238
High Energy Astronomical Observatory, 43, 55, 56
High Energy Astronomical Observatory-2, 43
Hubble Space Telescope, 27, 55–59, 64, 132, 228, 232, 253
International Ultraviolet Explorer, 49, 55, 141
James Webb Space Telescope, 49, 54, 63, 135, 136, 143, 232
Lander Amorphous Rover Antenna (LARA), 214
Landsat, 61, 63, 140
Laser Interferometer Space Antenna (LISA), 8, 46, 65, 230
Magnetospheric Multiscale, 7, 193
Magnetotail Constellation, 9, 15, 194
Medium-Class Explorers, 52, 58, 230
Microwave Anisotropy Probe (MAP), 14, 15
Orbiting Astronomical Observatory-3, 55
Orbiting Solar Observatory-8, 56
Prospecting Asteroid Mission (PAM), 213, 214, 216, 221
Rossi X-ray Timing Explorer (RXTE), 54, 59–62, 64, 126, 238
Saturn, 219
  Titan, 219, 231
Saturn Autonomous Ring Array (SARA), 214
Small Explorer, 31, 61, 121
Sojourner, 18
Solar Anomalous and Magnetospheric Particle Explorer, 61
Solar Dynamics Observatory, 63
Solar Imaging Radio Array, 230, 231
Solar Maximum Mission, 55–58
Solar-Terrestrial Relations Observatory, 6, 7
Space Technology 5, 5, 15, 191
Stellar Imager, 230, 231
Swift, 16, 60, 62, 64, 130
  Burst Alert Telescope, 62
Techsat21, 203
Titan Explorer, 231
Total Internal Reflection Optical System, 193
Tracking and Data Relay Satellites (TDRS), 8, 51, 54, 60, 62, 63, 65, 121, 198
  Demand Access System, 62
Triana, 63
Tropical Rainfall Mapping Mission, 60, 61
Two Wide-angle Imaging Neutral-atom Spectrometers, 8
Upper Atmosphere Research Satellite, 58
Venus, 219
Willsinson Microwave Anisotropy Probe, 62, 64, 125, 238
mobile
  code, 228
  computing, 187
  software, 186
model checking, 223
modeling
  look-ahead, 120, 125, 127, 128, 140, 141, 143, 244, 245, 250, 251
  statistical, 183
modelwebs, 234

monitoring and trending, 122, 129
moon, 27, 126, 214
  outpost, 232

nanotechnology, 225
NASA, 3–5, 9, 13–15, 37, 38, 55, 69, 86, 111, 125, 140, 145, 147–149, 163–166, 169, 172, 175, 185, 188, 191, 194, 211, 213, 214, 216, 217, 219, 224, 225
  Goddard Space Flight Center, 38, 39, 42, 52, 55, 61–63, 65, 116, 121, 128, 213
  Institue for Advanced Concepts, 219
  Langley Research Center, 213
  New Millennium Program, 5, 62, 169
  research and development, 37
  Standard Spacecraft Computer, 55
  strategic plan, 227, 232, 234
natural language, 76–78
Navy, 224
networking, 19, 113, 182, 187, 193, 195, 200
  ambient, 186
  wireless, 186
neural networks, 106, 107, 118, 122, 123, 127, 169, 187, 220, 221
Nomadic Computing, 187

object-oriented
  design, 35, 137
  development, 222
on-orbit sparing, 196
operations concept, 134
operations research, 182
optimization, 38, 148
  combinatorial, 220
  network, 220
  query, 181
orbit, 26, 237
  control, 237
  correction, 247
  determination, 25, 29, 33, 120, 121, 123, 125, 126, 129, 130, 139, 140, 143, 237, 238, 243, 247
  generators, 79
  geostationary, 49, 232, 234
  geosynchronous, 28, 49, 122, 138, 219, 249–252
    celestial pointer, 138, 141
    earth pointer, 138, 141
  interpolator, 237
  low earth, 27, 28, 41–43, 46, 136, 138–142, 192, 196, 234, 237, 246, 250–252
    celestial pointer, 138–140
    earth pointer, 138, 140, 246
    sun-synchronized, 249
  maintenance, 123
  maneuver, 33, 120, 122, 129, 130, 134, 139, 141, 143, 158, 202, 238, 244, 247
    planning, 33, 46
  medium earth, 232
  perturbations, 238
  prediction, 34, 237
  propagation, 55, 63, 237
  stationkeeping, 20, 29, 46, 122, 136, 139–141, 238, 246, 247
  time processor, 58

persistence, 72
Personal Computing, 187
pervasive computing, 186
planner, 85, 98
  architecture, 97
  case-based, 101–103
  decision-theoretic, 70
  model-based, 100, 101, 103
  numeric, 99
  symbolic, 98–100, 103
planning, 13, 96–98, 140, 145, 149, 152–154, 160, 166, 167, 202, 203
  command sequence, 79
  context, 99
  cycle, 163
  deliberative, 96, 154
  execution, 85
  globally optimal, 97
  hierarchy, 162
  high level, 169
  iteration time, 162
  mission, 13, 53, 156, 158–160, 162–164, 166, 169
  reaction time, 162
  reactive, 96, 99, 100, 154, 169
  replanning, 42, 97, 158
  science, 156, 159, 161–164, 166, 167

sequence, 156, 158, 160, 161
sub-optimal, 97, 101
target, 128
time, 162
planning and scheduling, 25–28, 35, 43, 46, 63, 66, 70, 80, 84, 85, 87, 88, 90, 91, 115, 119–123, 125, 127–130, 139–143, 163, 169, 238, 242–244, 247, 248, 250–253
architectures, 169
distributed, 197
science, 66, 123
power
fuel cell, 232
management, 33
nuclear, 232
solar, 232
process algegra, 224
progressive autonomy, 91, 138, 194, 205–207, 209
protocol
agent, 75, 86, 138, 188
efficiency, 114
Internet, 172
space, 28, 192
publish-and-subscribe, 83, 87, 184

race conditions, 223
radar, 109
Rapid Spacecraft Development Office, 65
reasoning, 82, 199
case-based, 30, 101, 102, 118, 122–124, 127, 130, 170
expert system, 87
heuristic, 218
model-based, 169
partial information, 104
probabilistic, 183
reflection, 12, 182
reflexive, 12, 182, 199, 202
routine, 12, 182
rule
development, 183
engine, 185
rule-base, 70, 71, 87, 187
reconfiguration, 234
Recovery Oriented Computing, 182
redundancy, 47, 61, 185, 193, 195, 196, 211
reengineering, 25, 26
reliability, 196, 228
remote sensing, 234
resource
constraints, 192
limited, 208
management, 39, 44, 49, 50, 128, 134, 137
robots, 3, 17–19, 22, 23, 26, 99, 113–115, 144–146, 149, 154, 169, 173, 175, 208, 228, 232, 234
actuator, 22, 109
cooperative, 18
exploration, 38
immobile, 23
mobile, 169
navigation, 23, 144
reactive control, 22
sensors, 22
Sojourner, 18
space-based, 18
underwater, 170
robustness, 185
root-cause analysis, 183

Saturn, 214, 231
scheduling, 103, 125–134, 139, 140, 163, 232
absolute-time, 135, 136, 139
adaptive, 132, 135–137
calibration, 128, 131
dynamic, 126, 128
event-based, 135–137, 139
generation, 41
goal-driven, 130, 131
ground, 41, 128
onboard, 49, 140
opportunistic, 131
predictive, 162
rescheduling, 43
science, 26, 49, 128
short-term, 137
slective target, 43
spacecraft, 41
target, 128
Schetter, Thomas, 202
science

coordinated, 193, 196
data, 43, 59, 148
data processing, 31, 33, 34, 115
execution, 39, 41, 132
goals, 156
negotiation, 158
observation, 50, 129, 132, 247
opportunistic, 16, 27, 38, 54, 56, 59, 60, 62, 63, 125–131, 137, 139, 140, 143, 162, 209, 228, 242, 244
optimization, 38
planning, 13
schedule, 28
support activities, 26, 28
targets, 27, 58
semantics
declarative, 171
grammar, 76
sensor
calibration, 33
distributed, 185
network, 18, 23
sun, 238
web, 232, 234
sigma-editing, 122, 127
signal processing, 109, 110
simulation, 111, 112, 122
distributed interactive, 114, 172
environments, 113
forces, 172
hardware, 112
networked, 113
onboard, 249
servers, 113
SimNet, 172
vehicle, 172
software
cost, 51, 54, 143
development, 182
engineering, 176, 178, 179, 188
hot swapping, 181
solar
array, 21, 45, 231
flare, 56
magnetic activity, 230
position, 243
sail, 217, 221, 225
wind, 5, 7
sonar, 109
South Atlantic Anomaly, 27, 43, 59, 60, 121, 123, 247, 248
space physics, 194
space-based processing, 192
spacecraft
anomaly, 32, 35, 39, 42, 45, 52, 53, 62, 80, 81, 90, 117, 119, 122, 123, 128, 131, 134–137, 142, 158, 195, 197, 208, 209, 245, 250, 253
antenna, 54, 59, 60
aperature, 201
attitude
actuator, 239
control, 21, 25, 29, 31, 32, 35, 43, 50, 54, 56, 58, 61, 117, 120, 121, 125, 133, 134, 139, 141, 143, 232, 237, 238, 243, 244
determination, 10, 29, 33, 55, 56, 119, 120, 125, 129, 130, 139, 140, 143, 237, 238
error, 238
maneuvering, 129, 130, 132
measurement, 237
sensor, 140, 238, 239
autonomy, 38, 206
battery, 45, 54
bus, 33–35
calibration, 25, 26, 31, 56, 66, 117, 119–121, 125–129, 131, 132, 139, 140, 142
celestial pointer, 28, 42, 45, 50, 117
commands, 41
absolute time, 41, 42, 49, 55, 130, 137
conditional, 41–43, 55
configuration, 141
delta-time, 42
execution, 44, 48, 49
loading, 26, 28, 66
management, 34
processing, 33, 34
real-time, 43, 137
relative-timed, 41, 42, 55, 58
scripts, 244
sequencing, 158, 166
timeline, 42
timing, 253
validation, 48, 247
verification, 161

communications, 21, 231
computer, 31, 33, 46, 55, 59, 63, 116, 125, 237, 253
  DF224, 57, 58
  NSSC-I, 55, 58
design, 3, 65, 145
engineering support, 26, 28
ephemeris, 19, 20, 27, 52, 54, 58, 60, 65, 121, 123
guidance, 57, 59, 136
Guidance, Navigation, and Control, 238
guide star, 57, 59, 64, 136
gyro drift, 54, 56, 121, 238
hardware, 41
inertial
  fixed pointing, 116, 117
  hold, 35, 59, 117, 120, 121, 135
instrument, 28, 31
  adjustment, 247
  calibration, 25, 28, 31, 53, 61, 66, 117, 120, 121, 129, 133, 139, 140, 143, 242, 243
  commanding, 33, 34, 116, 120, 124, 125, 128–130, 139, 140, 143, 243, 247, 248
  communications, 247, 251
  configuration, 42, 43, 63, 65, 120, 124, 125, 137, 139, 140, 143, 243, 247
  data storage, 247, 251
  diagnostic, 119
  Narrow Field, 63
  performance, 245
  pointing control, 43
  reconfiguration, 247, 248
  safemode, 35
  smart, 132
  survey, 64
intelligence, 202
launch, 196
mass storage, 126
microsatellites, 5, 231
mirrorsats, 230
momentum
  angular, 46, 49, 135
  dump, 29, 57, 64, 135
  management, 34, 46, 58, 117, 145, 241, 242
monitoring, 69
multi-, 14, 15
nano-class, 9, 89, 192–194, 201
operating system, 126
operations, 14, 20, 21, 26, 37, 38, 40, 43, 46, 48, 51, 52, 134, 169, 202
Optical Telescope Assembly, 28, 34, 133
orientation, 237
performance monitoring, 26, 29, 66
pico-class, 211, 214, 216, 219, 225
pointing control, 43, 44, 55, 57
power
  electrical, 45, 134
  management, 21, 34, 36, 45, 55, 116, 118, 145
processing, 34, 67, 247
propulsion, 21, 117
  management, 33, 46
  propellent, 134
  subsystem, 34
reaction wheel, 46, 239, 243
safemode, 195, 245
schedule, 41
simulations, 132
spin-stabilized, 35, 59
state, 53
storage, 34, 60, 237
subsystems, 208
support activities, 66
support functions, 27
systems access, 51
telemetry, 10, 29, 30, 32, 34, 40, 44, 45, 50–53, 55, 57, 58, 60, 62, 73, 80, 81, 90, 134, 143, 144, 157, 158, 160, 194, 195, 197, 245, 251
  filter table, 53
  formats, 53
  monitor, 52, 57, 58, 61, 194
thermal management, 21, 33, 34, 36, 50, 54, 116, 118
thruster, 34, 117
  control, 115, 122
  firing, 116
  management, 117, 134, 238
  optimization, 57
uplink-downlink card, 35
standard deviation, 122, 127
star catalog, 238

star tracker, 10, 55, 61, 120, 125, 126, 238, 239
  lost in space, 62, 238
  quaternion, 62, 243
state modeling, 30, 118, 122, 123, 127
state-based systems, 199
state-space, 223
stochastic methods, 187
strategy-schema, 71
string of pearls, 191
Sumpter, David J.T., 224
sun
  acquisition, 35
  centroid, 63
  coronal mass ejections, 231
  point, 55
  radio bursts, 231
sunpoint, 59
Super Miniaturized Addressable Reconfigurable Technology, 214
Surka, Derek, 202
swarm, 14, 175, 201, 211–225
  Ant Colony Optimization, 220
  applications, 212
  autonomic, 220
  birds, 220
  boids, 220
  definition, 212
  insect behavior, 219
  intelligence, 212, 213
  optimization, 212, 220
  particle, 220
  quantitative structure activity relationships (QSAR), 220
  robotics, 213
  simulation, 212
  Super Miniaturized Addressable Reconfigurable Technology (SMART), 213
  team, 217, 221
  tetrahedral walker, 211, 213–216
  verification, 222, 223
systems
  engineering, 178, 179
  management, 182
  of systems, 176, 184

target
  acquisition, 32, 36, 43, 58, 60, 125, 126, 132, 143, 243, 246
  identification, 43, 234
  observation, 43
  quaternion, 59, 63
targeting, 130
task management, 33, 34
tasks, 98
TCP/IP, 74, 78
terrestrial databases, 195
testing, 95, 110–112, 114, 222, 223
  environments, 112, 113
  plan, 110
  simulation, 111, 112
  software, 172
  time, 112
time management, 33, 34
total cost of ownership, 176, 188
tracking station, 195
trajectory
  planning, 202
traveling salesman problem, 220
trend analysis, 195

UK Computer Science Grand Research Challenges, 176
University of California Berkeley, 220
University of Maine, 170
Unmanned Aerial Vehicle, 211, 220
Unmanned Underwater Vehicles, 220
user modeling, 69
utility computing, 186

validation, 48, 132, 205, 206, 229, 244, 249
verification, 91, 205, 206, 222, 223, 225, 228, 229
  command, 161
  emergence, 223
  formal methods, 223
  learning, 223
  swarm, 223
Virginia Tech, 219
virtual
  environment, 172
  lens, 196
  platform, 147, 149, 164, 166–168, 191
  presence, 169
  telescopes, 193

world, 18, 19, 172
Visual Analysis Graphical Environment (VisAGE), 80

Weighted Synchronous Calculus of Communicating Systems, 224
Weiser, Mark, 186
Workplace, 86
world computing, 186

X-Machines, 224
X-ray, 230

Zadeh, Lotfi, 104