A Policy-Based Management Architecture for Mobile Collaborative Teams

Eskindir Asmare, Anandha Gopalan, Morris Sloman, Naranker Dulay and Emil Lupu
Department of Computing, Imperial College London, London, SW7 2RH, UK
Email: {e.asmare, a.gopalan, m.sloman, n.dulay, e.c.lupu}@imperial.ac.uk

***Abstract*—Unmanned Autonomous Vehicles (UAVs) are increasingly deployed for missions that are deemed dangerous or impractical to perform by humans. Collaborating UAVs need to adapt their behaviour to current context. They should be self-managing in that they have to recover or adapt to component failures and optimise performance to best utilise available resources. Our objective is to develop a framework which enables UAVs to manage themselves as an individual and as a team. We make use of policy-based techniques to support adaptive management. We use three levels of specifications namely, policy specification, mission class specification and mission class instantiation specification so that it enables us to reuse policies and the mission classes. The proposed architecture can be applied to systems with limited capabilities, such as devices in a Personal Area Network, as well as to systems with more capable devices. The evaluation of our proposed architecture shows that the architecture is scalable and also outperforms a centralised mission management scheme.**

I. Introduction

Technological advances in engineering and communication have paved the way for increasing use of autonomous systems. Given high level objectives from administrators an autonomic computing system is able to self-manage. It is also able to simplify the task of managing today's complex computing systems [1], [2]. Self-managing autonomous systems have the potential for providing the information needed to assist rescue operations - by locating survivors, identifying affected areas, and organising the collaborative efforts of the response team members. They also find uses in disaster management, including earthquakes, forest fires and foods, unmanned vehicles and military applications. In this paper, we focus on teams of unmanned autonomous vehicles, since this is the testbed that we have been using.

Unmanned Autonomous Vehicle (UAV) is a type of robot that due to its unique nature has various individual and collaborative applications (as mentioned above). Recently, there has been an increasing tendency to use UAVs in civilian disaster relief missions and military scenarios to reconnoiter or provide sensing in areas which are dangerous or impractical for humans. To achieve a particular mission such as surveillance of a specific area or search for specific targets, teams of UAVs may need to cooperate. A challenge in using UAVs for such missions is enabling adaptive self-management so that they can adapt to changes in context and accomplish a mission autonomously, i.e., without human intervention.

UAVs need to adapt their behaviour to current context - location, activity, available resources such as battery power and available services such as quality of communication links. They should be self-managing in that they have to recover or adapt to component failures and optimise performance to best utilise available resources. Collaborating UAVs form a Self-Managed Cell (SMC) [3], which is the general architectural principle we use for realising self management of individual and groups of UAVs. The SMC is an approach to support autonomic computing [2]. A SMC group tend to consist of multiple UAVs and at least one commander, which could be a human or a UAV, to effectively control the group. There could be back-up commanders in case the primary one fails. The SMC is set up to perform a mission based on the mission specification received by the commander from its command base. The mission specification defines how UAVs will be assigned to perform specific roles within the SMC, based on their credentials and capability description. It also defines as to when and how to adapt the mission to changes in context or failures. Multiple teams of UAVs may also collaborate to achieve the overall mission. SMCs are defined so as to allow the composition of SMCs to form larger and more complex SMCs.

UAVs need to manage themselves as an individual and as a team. Management of UAVs involves resource, task, behaviour, communication and team management. The adaptive management of UAVs is achieved by using policy-based techniques that allow dynamic modification of the management strategy without reloading the basic software within the UAV. A UAV contains a capability description that describes its resources and the services it can perform. A UAV may be composed of various sensors for vision, sound, vibration, chemical detection, location and devices for communication. Usually, not all capabilities are available in a single vehicle and so one UAV may provide services to others in the team.

Although there have been various research on control architectures for autonomous systems, the focus has largely been in organising intelligence. We argue that if robots such as UAVs are to be used in real life applications then they should also be able to manage their intelligence. The focus of this paper is distributed self-management of a group of UAVs. We use the Ponder2 [4] policy framework in our management architecture. It is a generic object management system which allows the dynamic loading, unloading, enabling and disabling of managed objects. Generally a Ponder2 managed-object is an active object that is capable of receiving action commands and performing actions. Ponder2 supports two types of policies namely obligation and authorisation policies. We use obliga-

tion policies (event-condition-action rules) to trigger specific actions to be performed when an event, such as the discovery of a new UAV occurs, or a UAV which is a member of the team fails. Authorisation policies are used to specify what services and resources within a UAV can be accessed by other UAVs performing a specific role and under what conditions. Policies are interpreted; as a result they can be dynamically loaded, enabled or disabled at run-time without shutting down a system in order to adapt the management strategy being used within the system. To summarise, our management framework addresses mission management, capability description and distributed control and communication.

The rest of the paper is organised as follows. Section II presents the distributed mission management architecture. Section III details the mission specification, while Section IV details the implementation of the proposed architecture. Section V details the experiments and the ensuing results. Section VI details the work related to this paper, while Section VII concludes the paper and provides ideas for future work.

II. Distributed Mission Management

A mission for a team of heterogeneous robots is specified in terms of roles and role-missions. There are various classes of mission specifications. We broadly divide these classes into three: (a) the domain they are targeted to: application specific [5] or generic [6], (b) the paradigm they use: plan based [7] or specification (configuration) based [6], and (c) the number of autonomic systems involved: e.g. single or multi-robot missions. One can further divide these classes, for e.g. a multi-robot mission could support homogeneous or heterogeneous robots and a heterogeneous-multi-robot mission could support static or dynamic task allocation. Based on the above classes of mission specifications, the mission specification that we use is: generic, specification based, multi-robot and heterogeneous with dynamic task allocation.

To illustrate our approach, we consider an example mission consisting of determining whether an area is safe to be entered by humans. The following main roles are identified: *Commander (C)*: controls the mission and allocates UAVs to roles. *Surveyor (S)*: explores the house and builds a map. *Hazardous material detector (H)*. *Communication relay (R)*: maintains communication among UAVs by forming an ad-hoc network. *Aggregator (A)*: aggregates information from all UAVs e.g. to produce a map showing the detected hazardous materials.

A. Our Approach

A team of UAVs should be able to perform a mission with a minimum number of UAVs with required capabilities although the configuration may not be optimal. When additional UAVs become available the team should expand to make use of the new resources, thereby increasing performance. Should there be a failure or departure of UAVs from the enlarged team, the team should contract but continue the mission. We define a *minimal team configuration* as the fewest types and number of UAVs needed to accomplish a mission. A reasonably-optimal team configuration has the ideal type and number of UAVs. A mission starts execution when a team satisfying the minimal configuration can be formed. The team will expand when additional UAVs join until it achieves the reasonably-optimal configuration.

This concept is illustrated using the reconnaissance scenario in which the minimal configuration is one Commander, one Aggregator, and two Surveyors, where the Surveyor role is the primary role; and the Relay and Hazard detection roles are secondary roles. The primary role is given priority with respect to resource allocation and is also responsible for managing the other roles of the UAV.

As shown in Fig. 1, the Surveyor role is collocated with other roles: Relay and Hazardous material detector. The Relay role can be performed in parallel with either the Surveyor or the detector role while the UAV has to switch between the Surveyor and detector roles as only one of these can be active at a time. Although the Relay role can run in parallel with these roles, it will potentially hinder the surveying or detection missions when trying to maximise communication link quality. Hence this role is best placed in another UAV should there be one available. Detecting potentially hazardous objects is a much slower process than exploration which implies that the detection and surveying missions can be performed better by separate but cooperating UAVs which share information.

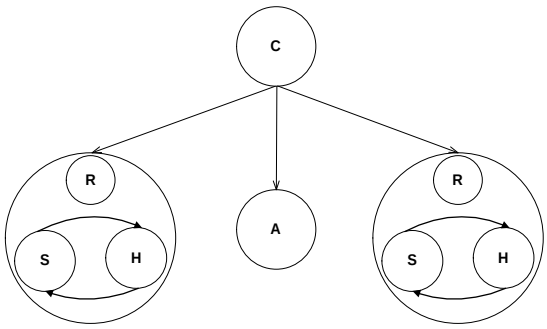

Fig. 1. Reconnaissance mission minimal configuration

A reasonably-optimal mission configuration consists of one Commander, two Surveyors, two Hazardous material detectors, two Relays, and one Aggregator. The team started with the configuration shown in Fig. 1 and reached the configuration shown in Fig. 2 as new UAVs join the team in the mission area. The Surveyor roles which assigned the Relay and detector roles serve as commanders for those two roles. Should any of the new UAVs fail or depart, the roles will revert to their minimal configuration position.

B. Conceptual Model

In this section, we present our model that uses the concepts of team-mission, role and role-mission that allow for distributed mission management of UAVs.

1) Team Mission (M): A team-mission is a set of roles where each role contains a set of policies that either governs the behaviour of the role or handles the assignment of UAVs to roles. $M = \{R_1,\ R_2,\ R_3,\ ...\ R_i\}$.

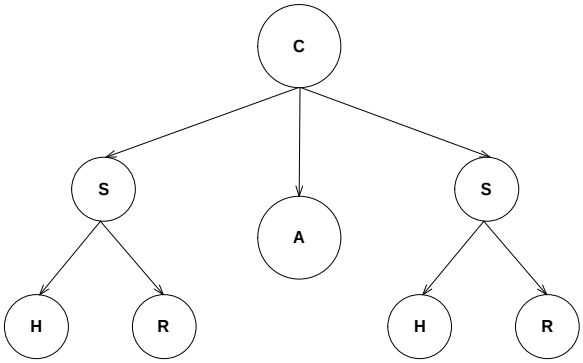

Fig. 2. Reconnaissance mission reasonably-optimal configuration

2) Role (R): A role is a placeholder to which discovered UAVs are assigned and for which, missions, tasks and authorisation policies are specified. The role is implemented as a set of three objects: role-mission (RM), authorisations (A) and tasks (T), as described below. When a UAV is assigned to a role the role-mission, authorisations and tasks associated with the role are loaded onto the UAV, unless already present. $R \supset \{RM,\ A,\ T\}$.

A role in our model, has an external and a local interface which provides a context for which role mission policies can be specified. Incoming events from the local or external interface can be used to trigger policies which invoke operations provided by the local and external interfaces.

2.A) External Interface: The external interface defines operations and events relating to interaction with external collaborating roles: (i) Management operations for loading missions, policies etc. which are common to all roles, (ii) Operations from the local interface that are made visible to and can be invoked by other roles i.e. other UAVs. Accessibility may be static or dynamic depending on the authorisation policies. The operations are implemented by the tasks in the role, (iii) Events generated from within the role, by the tasks inside the role or propagated from the UAV components such as sensors, and published via an event bus for use by other roles, (iv) Operations which are required by this role. These operations are expected to be provided by collaborating roles, and (v) Events required by this role, generated by collaborating roles, e.g. to trigger policies.

2.B) Local Interface: The local interface defines the operations and events provided by tasks within the local UAV and used by the role-missions: (i) Events generated by the tasks within the UAV or propagated from the UAV components such as sensors. These may be used to trigger policies in the mission or mapped to the external interface, and (ii) item Operations implemented by the tasks within the UAV.

3) Role-Mission (RM): A set of policies relating to a single role that allows for controlling tasks and enabling/disabling other policies.

4) Authorisations (A): The authorisation policies specify how roles are permitted to interact with each other in terms of the events that can be triggered or operations that can be invoked via the external interface. $A = \{A_1,\ A_2,\ ...\ A_j\}$.

5) Tasks (T): Tasks are complex operations which the UAV can perform e.g. move from A to B, follow a path, track an object using video. Obligation policies in the mission may invoke operations supported by a task or activate a task. The tasks in a role are usually inherent to the type of the role and hence are specified inside the role class. $T = \{T_1,\ T_2,\ ...\ T_k\}$.

III. Distributed Mission Specification

We use three levels of specifications, as shown in Fig. 3: (a) Policies are specified using the Ponder2 [4] policy specification language and stored in a policy repository, (b) the XML mission class specification of the types of roles needed for the mission, the policies required from the policy repository, and (c) the XML mission instance specification which defines the mission parameters and role cardinalities required to instantiate a mission class. The policy specification in the repository may apply to multiple mission classes and there can be multiple instances of a mission, instantiated with different parameters from a particular mission class. The policy repository only contains policies and hence does not require a lot of memory. When using the architecture for less capable devices such as PDAs, mobile phones etc., the repository can be implemented in memory by the initial commander UAV and distributed to other UAVs over time. In the proof of concept demonstration (Section IV-C), the policy repository was maintained by the commander, which was a laptop connected to the other UAVs through the ad-hoc network.

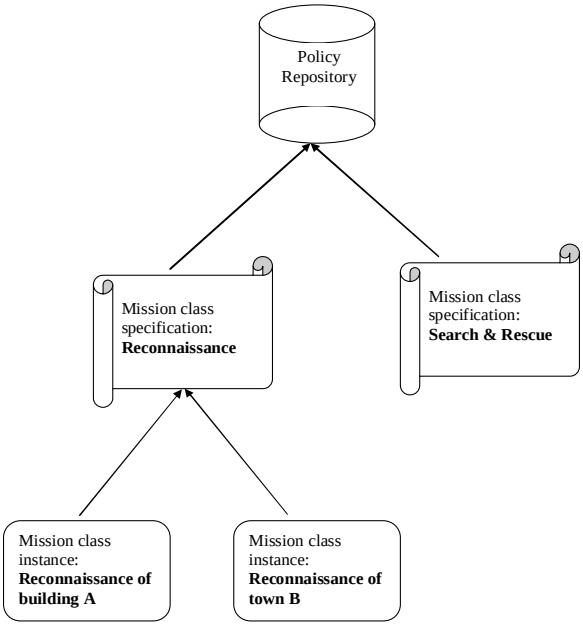

Fig. 3. Mission Specification Levels

In the following sections, we describe the specification schemes in a bottom-up fashion. We start from policies which are the units of the mission specification and go up to

mission class instantiation which is the highest level of the specification. Section III-A and Section III-B describe policy and mission-class specifications respectively and Section III-C describes mission class instantiations.

A. Policy Specification

The policies specified for a role are broadly divided into role assignment policy and operational management policy. Role assignment policies are used to assign UAVs to roles based on their capabilities. Operational management policies are used by roles to manage their own operational behaviour or operational behaviours of collaborating roles.

Table I shows an example of a role assignment policy. In this policy the commander checks the capability of a newly discovered UAV and authenticates it and determines whether it can assume the role of a surveyor; if the capability is satisfied, the new UAV is given the role of a surveyor. Table II shows an example role re-assignment policy.

TABLE I
PONDER2 POLICY FOR ROLE ASSIGNMENT

```
policy event: /event/newUAV;
condition:[:lowLevelCap :uav|
    root/discovery uav: uav  has:
    #("motion" "video") cap: lowLevelCap
    auth: credential];
action: [ :name :uav |
    root print: "Received event:" + name.
    root print: "Checking low level capability".
    root/discovery fullCapReq:uav role:"surveyor"].
```

TABLE II
PONDER2 POLICY FOR ROLE RE-ASSIGNMENT

```
policy event: /event/UAVFailure;
condition:[:role| role=="surveyor"];
action: [ :role :name |
    root print: "Received event:" + name.
    root/role/commander reassign:role
    scheme:"default"].
  root/policy at: "reAssignSurveyor" put: policy.
```

B. Mission Class Specification

Mission classes can be specified by describing what policies a role uses to manage itself and others (where hierarchy exists). A mission class effectively specifies a team by using roles and showing the management relation among the participating roles as well as the cardinality of each role. A mission class specification for the reconnaissance scenario is provided in Table III. In this specification, we have a commander role managing a surveyor and an aggregator role, where the surveyor role in-turn manages an hDetector and relay roles. The cardinality and other parameters are instantiated later. Mission parameters such as failure-timeout which are shared by all roles are also included in this specification. This specification can be used to instantiate different teams of the same configuration with different cardinalities, mission parameters and role behaviours.

TABLE III
MISSION CLASS SPECIFICATION

```
<!--Mission Class Specification-->
<xml>
<missionClassParameters>
    <missionClassName>Reconnaissance
    </missionClassName>
</missionClassParameters>
<constraints>
    <cardinality/>
    <colocation/>
</constraints>
<missionParameters>
<!--A mission class instance should provide values
    for these parameters-->
    <comTimeout> int</comTimeout>
    <failureTimeout>int</failureTimeout>
</missionParameters>
<commanderBehaviour>
    <roleManagement>
        <manages>surveyor</manages>
        <manages>aggregator</manages>
    </roleManagement>
</commanderBehaviour>
<surveyorBehaviour>
    <cardinality>int</cardinality>
    <roleManagement>
        <manages>hdetector</manages>
        <manages>relay</manages>
    </roleManagement>
</surveyorBehaviour>
<aggregatorBehaviour>
    <cardinality>int</cardinality>
    <roleManagement/>
</aggregatorBehaviour>
<hdetectorBehaviour>
    <cardinality>int</cardinality>
    <roleManagement/>
</hdetectorBehaviour>
<relayBehaviour>
    <cardinality>int</cardinality>
    <roleManagement/>
</relayBehaviour>
</xml>
```

C. Mission Class Instantiation

A mission class instance (which gives rise to the actual team of UAVs performing the mission) specifies values for cardinalities, mission parameters and URIs of policies which define the role behaviour. Table IV shows a mission class instance specification for the reconnaissance scenario mentioned previously.

D. Management Tree

As a means of decentralising discovery and role management, the UAVs in a mission are arranged in the form of a management tree during the role assignment process. This tree is used for defining management hierarchies as well as data aggregation during execution of the mission. In this section we present a simplified version of the algorithm used to form the management tree.

Each UAV runs the tree formation algorithm which starts by broadcasting a discovery message. UAVs receiving the discovery-broadcast perform an authentication protocol if they

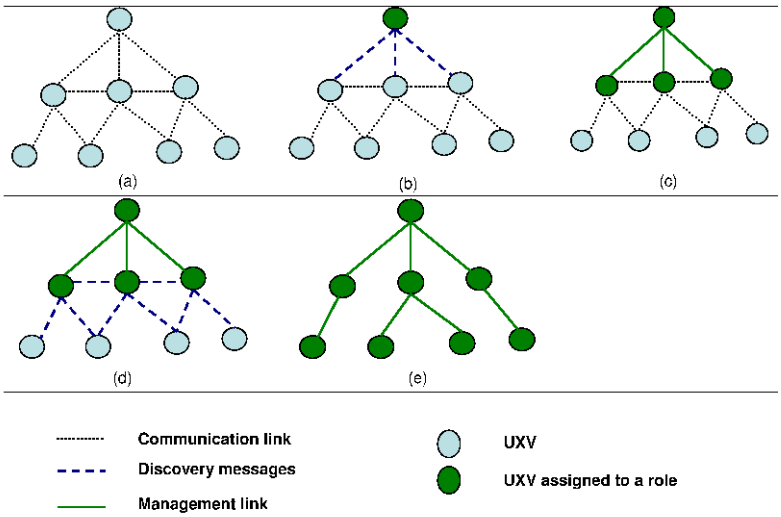

Fig. 4. Management Tree Formation

are not already a member of the SMC and reply with a summary of their capability description if they can be assigned to a role, i.e., UAVs which are already assigned to a role may ignore the broadcast message. Upon authenticating and receiving the capability summary, the broadcaster decides whether to request for a full capability description. The authentication protocol is based on the use of public keys and will not be described in this paper. The final decision of assigning the UAV to a role takes place after checking the full capability description against the requirements of the role. If the UAV is assigned to the role the broadcaster will be the parent for the UAV that replied and the UAV will be listed as a child of the broadcaster (shown in lines 6 and 15 of Algorithm 1). The full steps are shown in Algorithm 1. Algorithm 1 makes use of Algorithm 2 for performing the role assignment.

Fig. 4 illustrates a trace of the protocol in action. Fig. 4 (a) shows the communication links between neighbouring nodes. In Fig. 4 (b) the top node broadcasts Discovery messages to its neighbours which eventually form a team with the top node as commander and the middle node as children assigned to various roles. In Fig. 4 (d), the middle nodes broadcast to their neighbours but only the lower nodes respond as the other middle nodes already have a parent. Fig. 4 (e) shows the resulting tree with each node having a single parent. The notations used in the algorithm are: *DISCOVERY_MSG*: requests the UAVs to send a summary of their capability as well as providing information, such as identity, about the UAV broadcasting the discovery message, *DISCOVERY_REPLY*: message containing a summary of a UAV's capabilities, *ROLE_ASSIGNMENT*: message containing role assignment information, $Parent$: manager of the UAV, C: list containing child UAVs, i.e., UAVs managed by the parent UAV, IS_ACTING_CDR: boolean value that shows whether this UAV is the acting commander for the mission and MSG: any message received by the UAV.

Algorithm 1 Management Tree Formation Algorithm

MGMT-TREE-FORM(*IS_ACTING_CDR*)

Input: IS_ACTING_CDR

1: Broadcast *DISCOVERY_MSG*
2: Receive *MSG*
3: Authenticate sender of MSG
4: **if** $IS\_ACTING\_CDR == TRUE$ **then**
5: **if** MSG != $DISCOVERY\_MSG$ **then**
6: Append ROLE-ASSIGN(MSG) to C
7: **end if**
8: **else**
9: **if** $MSG == DISCOVERY\_MSG$ **then**
10: **if** $PARENT == NULL$ **then**
11: Reply with capability summary
12: Handle further communications, if any
13: **end if**
14: **if** $MSG == DISCOVERY\_REPLY$ **then**
15: Append ROLE-ASSIGN(MSG) to C
16: **else**
17: **if** $MSG == ROLE\_ASSIGNMENT$ **then**
18: $PARENT$ = sender of MSG
19: **end if**
20: **end if**
21: **end if**
22: **end if**
23: **return**

Failure of a communication link and/or a UAV causes partitioning of the team network as well as loss of functionality; we use a systematically defined identity for UAVs to facilitate merging and re-joining of partitioned teams. The identity I of a UAV is defined as: I = $[M \mid H \mid S]$ where: M is the mission ID, H is the hierarchy level and S is a numbering

TABLE IV
MISSION CLASS INSTANTIATION

```
<!--Mission-class instantiation specification-->
<xml>
<missionParameters>
    <comTimeout>3000</comTimeout>
    <failureTimeout>7000</failureTimeout>
</missionParameters>
<commander>
    <cardinality>1</cardinality>
    <policyRepository>
    http://192.168.0.1/policy/commander
    </policyRepository>
</commander>
<surveyor>
    <cardinality>1</cardinality>
    <policyRepository>
    http://192.168.0.1/policy/surveyor
    </policyRepository>
</surveyor>
<aggregator>
    <cardinality>1</cardinality>
    <policyRepository>
    http://192.168.0.1/policy/aggregator
    </policyRepository>
</aggregator>
<hdetector>
    <cardinality>1</cardinality>
    <policyRepository>
    http://192.168.0.1/policy/hdetector
    </policyRepository>
</hdetector>
<relay>
    <cardinality>1</cardinality>
    <policyRepository>
    http://192.168.0.1/policy/relay
    </policyRepository>
</relay>
</xml>
```

Algorithm 2 Role Assignment Algorithm

ROLE-ASSIGN(MSG)

Input: MSG
Output: *CHILD*

1: Check capability summary
2: If the summary is viable, request for full capability description
3: If the description matches the requirement for one of the roles, do role assignment
4: Add the assigned UAV to *CHILD*
5: **return *CHILD***

system which helps to put all the UAVs in the management hierarchy in a total order. The identity lasts throughout the team configuration. It identifies a mission and the hierarchy level of a UAV in a management tree. The ability to identify this level is useful in handling intermittent link disconnections. Due to the lack of space, failure management will not be explored further in this paper.

Using the management tree algorithm described above, the management tree for the mission class specification in Table III will look like the tree in Fig. 5.

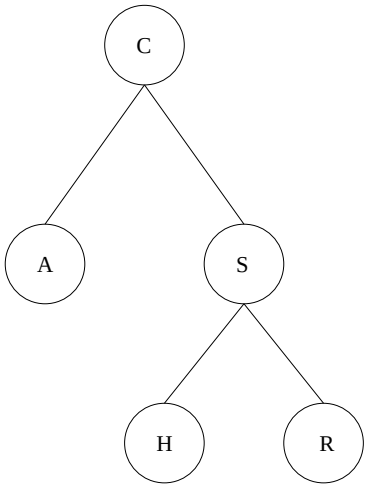

Fig. 5. Initial Management Tree

IV. IMPLEMENTATION

A. Mission management

The management architecture is implemented using the Java based Ponder2 policy toolkit with Tasks, Roles and Missions implemented as Ponder2 managed objects. An outline of the framework is shown in Fig. 6.

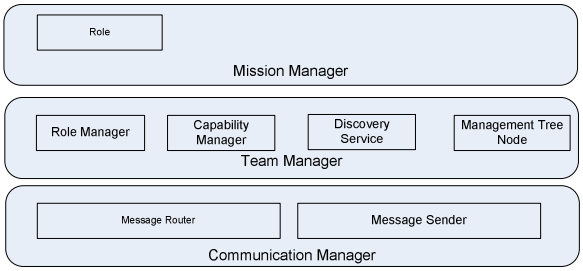

Fig. 6. Mission Management Architecture

We use a connectionless communication mechanism for most of the messages exchanged in our system and hence the Communication-Manager is build upon the UDP protocol. However, although a connectionless communication is sufficient enough for most of messages exchanged between the UAVs, for critical messages such as role assignment and capability description messages a reliable delivery is necessary. The Message-Sender object serves this purpose.

The Message-Router handles messages for multiple roles (and other objects) residing in a UAV. This object enables registration of roles to receive packets of a certain type and/or source as well as de-registration. In the case when a role registers to receive packets of more than one type (or source) that intersect, the registrations are aggregated. A separate exclusion table is kept when a role de-registers if that role and the registration it is de-registering to lies in an aggregate registration. When a new packet arrives the dispatch table as well as the exclusion table are checked before the packet is passed to the registered roles.

The Role-Manager object is responsible for loading and withdrawing a role in mission startup and reconfiguration respectively.

B. Capabilities

The capability of a UAV is the set of operations which the software and hardware in the UAV support as well as the events it generates. This depends on the current set of software tasks loaded into the UAV. The capability specification of a UAV is generated dynamically by querying the tasks in a UAV. The Capability module is responsible for querying and preparing the description. Tasks support reflection so they can be queried for their interface description. The Capability module queries each task and produces the description based on the reply it gets. A task implements a task interface with a naming scheme where the interface name is the task name suffixed by an 'I'. For instance, an *Explorer* task implements an interface called *ExplorerI*. To facilitate the capability description generation, we annotate task interfaces using two annotations, namely *@TaskInterface* and *@TaskEvent*. The *@TaskInterface* annotation is used to mark (and indicate the name of corresponding task) that an interface is a task-interface in that it has operations/events that can be included in the capability description. This marking is used later on while generating the capability description to differentiate between the different interfaces a task implements. The *@TaskEvent* annotation is used to mark events so that we can differentiate between the tasks operations and its events.

As discussed in the previous section, tasks extend the Task class which supports our algorithm for capability description generation of a single task. The algorithm reads the interfaces implemented by the task reflectively and decides whether to consider the interface in general or the methods of the interface in particular by using the annotations. The pseudo code of this algorithm is shown in Table V.

TABLE V
PSEUDO CODE OF REFLECTIVE CAPABILITY DESCRIPTION GENERATION ALGORITHM

```
Determine, reflectively, all interfaces implemented
by the task
FOR all interfaces implemented by the task
Check for a @TaskInterface annotation
IF the annotation is detected THEN
  Check if the task matches the task indicated
  in the annotation
  IF it matches THEN
    FOR all methods in the interface
      Check for a @TaskEvent annotation
      IF the annotation is detected THEN
        Add the method to the description as
        an event
      ELSE
        Read the argument types reflectively
        Add the method as an operation
      ENDIF
  ELSE
    Return error
  ENDIF
ELSE
  Return empty description
ENDIF
```

C. Proof of Concept Demonstration

The distributed mission management architecture detailed in Section II was implemented on the Koala robots [8]. The Koala robot is a mobile robot which has 16 infrared proximity sensors around the body of the robot, and a camera. The Koala robot is controlled by an Asus EEE PC running windows and Java through an USB to serial cable. The scenario chosen for the demonstration was a search and rescue mission of a wounded soldier. The soldier is assumed to posses a wearable computer and a body sensor network that monitors the soldier's condition. The wearable computer was another Asus EEE PC, while the commander was a laptop and two robots were used as the unmanned vehicles. None of the wireless devices were connected to the infrastructure and all the devices were part of the same ad-hoc network.

The steps are as follows: (i) Soldier is wounded in the battlefield, (ii) Wearable computer sends a distress signal to the base reporting on the soldier's condition, (iii) The Commander assembles the mission for assistance, comprising unmanned vehicles capable of navigation, communication, surveillance and hazard detection. In this scenario, two roles are assigned: the surveyor and the aggregator, (iv) The surveyor robot starts to move towards the soldier and detects a hazard along the way, (v) On detecting the failure (through timeouts), the surveyor role is re-assigned to the aggregator by the commander. Also, the last position of the previous surveyor is relayed so that the new surveyor can avoid the "hazard", (vi) The new surveyor is able to avoid the hazard using the information provided, (vii) The new surveyor reaches the soldier and delivers assistance, as necessary. This proof of concept demonstration was shown as part of the Annual Conference of the ITA, 2008 (ACITA 2008) [9]. A snapshot of the demo is shown in Fig. 7. The first surveyor robot stops on detecting the "hazard" (which is a yellow cylinder). The second surveyor (which used to be the aggregator before the role re-assignment) avoids the obstacle and reaches the soldier.

Fig. 7. Snapshot of Proof of Concept Demonstration

V. Experiments and Results

In this section, we will detail the experimental setup, the experiments and the ensuing results while analysing our distributed mission management architecture.

A. Experimental Setup

The implementation of the distributed mission management architecture was carried out using Java based Ponder2 policy toolkit (Section IV). This allowed us to use a testbed that was made up of generic Linux machines running Java. The set of tests were conducted to find out the effect of our distributed tree management system versus a centralised scheme. We chose to conduct these sets of tests because they test the ability of the architecture to scale and also test the advantage of using a distributed mission management architecture. At the beginning of the simulation the number and types of roles are changed in the mission class specification and the requisite number of SMCs are started on various machines.

B. Results

The depth of the tree for this mission is assumed to be *4*. We measure the time taken for the mission to be started (which includes the UAV discovery, role assignment, downloading the policy from the repository and loading the policy and the time taken to start the role). The result of this experiment is depicted in Fig. 8. From the figure, we can see that when the number of roles is relatively small, the centralised scheme works best. Once the number of roles increases, the distributed scheme significantly outperforms the centralised scheme. Another fact worth noting is that the time taken for mission setup does not increase very much, even when the number of roles required in the mission is high. This augurs well for our architecture, since it shows that the architecture is scalable.

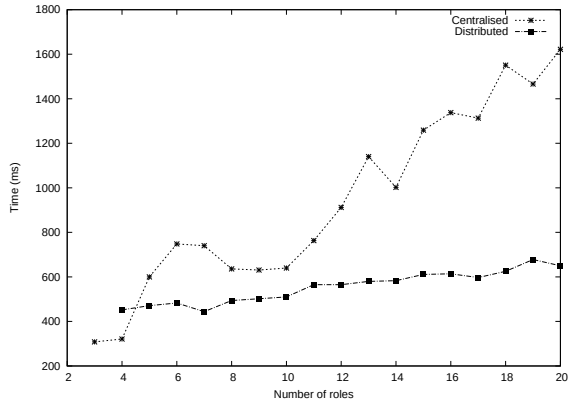

Fig. 8. Measure of time taken for mission setup versus the number of roles

VI. Related Work

Mackenzie et. al [6] present a mechanism for mission specification by specifying the organisation or set up of a set of primitives to obtain a sophisticated system which can perform complex tasks. The authors have developed a language called Configuration Description Language (CDL) and one can use this language to specify the configuration of a robot or group of robots. The configuration of a group of robots is the specification of the components, connections and structure of the control system. The lowest level this language goes to is calling the primitive modules, otherwise it does not specify how an actual action is performed. This makes the language robot-implementation independent.

The mechanism is developed for behaviour based robots. The authors define an agent as *a distinct entity capable of exhibiting a behavioural response to stimulus*. Using this definition enables mapping each primitive capability of a robot to an agent. These agents are called atomic agents. The authors then recursively define assemblage agents as *a coordinated society of agents*. The agents could be atomic or assemblage. Coordination determines how the society behaves (i.e. how it will react to a stimulus). The coordination can be competitive, temporal sequencing or cooperative. In competitive coordination a subset of the society is selected to do the activation. The selection is based on some metrics. In temporal sequencing a Finite State Machine which uses each agents behaviour as a state will be constructed. The behaviour of this machine is then the behaviour of the society. In cooperative coordination each agents behaviour will be assigned a vector and weight then the vector sum represents the society's behavioural consensus.

Using CDL a designer can define assemblage agents for different tasks and instantiate the primitives. Agents can be reused for a different task. The authors have developed a development environment (tool) called MissionLab which enables writing CDL and compilation. The executable can be loaded to a robot or a simulator. Also the tool enables graphical design of a mission and simulating the mission. However, the approach used to describe a mission, finite state machine is suitable only for low level components such as tasks and not easy for specifying multi-robot missions with many participants.

In [10], the authors build on the work in [6] and present a mission specification system with a case based reasoning approach for generating mission plans and a Contact Net Protocol [11] based task allocation.

Iocchi et. al. [12] present an approach for coordination of robots based on dynamic role assignment. The architecture of the system is layered with a coordination protocol running on top of a communication protocol. The basis of the communication protocol is the publish-subscribe paradigm. The coordination protocol is based on utility functions. A utility function is defined for each role. The robot with the highest value for a certain role will take that role. Formation is selected using a voting system. Compared to other works of coordination (e.g [13]), which tie the robot control architecture to the coordination architecture (mechanism), the authors' architecture is more general in that robots of different control architectures can coordinate.

In [11], the authors present the Contract Net protocol for distributing tasks through negotiation. Each node in the net takes one of the two roles, namely manager or contractor. Managers announce tasks, potential contractors submit bids to the managers, the managers then evaluate the bids and

award contracts to the bidders. The contents of the negotiation messages are problem-domain dependent and the user is responsible for specifying the content.

In [14], the authors present a paradigm for cooperating robots. In their approach, each robot has a hybrid automaton. Hybrid automata are used to represent roles, role assignments and discrete variables related to each robot. The composition of these automata is used to model execution of cooperative tasks. They define a role as a function one or more robots perform during the execution of a cooperative task and use utility functions to decide when to change roles.

Likhachev et. al. [15] have proposed an approach to automatic modification of behavioural assemblage parameters for autonomous navigation tasks. Their approach is based on Artificial Intelligence in that it uses case based reasoning. We try to solve a similar but more generic problem using a different approach, i.e., policy.

In [16], the authors present a general framework, called MURDOCH, for inter-robot communication and dynamic tasks allocation for cooperation. Communication takes place through a publish/subscribe system. MURDOCH offers a distributed approximation to a global optimum of resource usage. As a result of a completely distributed task assignment scheme, MURDOCH suffers the same problem as the greedy algorithms problem - equivalent to an instantaneous greedy scheduler, where decisions are made based on only the current and/or local situation without taking into account how the decision might affect the future and/or the global situation. In effect these types of algorithms may not always give the best solution. Although our approach does not take the global and/or future situation into consideration, in the window of time that every role assignment is done the architecture allows for optimisation on the set of discovered UAVs.

In [17], the authors present a distributed Constraint Programming (CSP) solution for assigning tasks to robots. They try to minimise remote task dependencies by creating a task dependency graph, which they call distributed organisational task network (DOTN), and search for minimal dependency solutions. The search for minimality is done during run time and it is done by trading tasks. They present an algorithm, called SOLO, for task reallocation.

Keoh et. al. [18] have proposed an approach to establish and manage mobile ad-hoc networks using policies. They introduced a community specification called *doctrine* which defines the roles of the participants in the community, the characteristics that participants must exhibit in order to be eligible to play a role, as well as the policies governing their behaviour within the community. However, while *doctrine* is focused on security our work deals with both security mission specification and task allocation.

VII. Conclusion

In this paper we have presented a distributed policy-based management architecture for mobile collaborative teams. We have also presented the models, concepts and implementation details for the management architecture. The management architecture is policy-based and uses three levels of specifications, namely, policy specification, mission class specification and mission class instantiation specification so that it enables us to reuse policies and the mission classes. A proof of concept demonstration of the proposed architecture was also achieved.

To ensure that the UAVs comprising the SMC perform their tasks correctly, it is important to cope with different types of failures. Also, it is sometimes desirable to make sure that the members of the team that form the SMC maintain communication links with the other members of the team. We will focus on failure and communication management as part of our future work.

Acknowledgements

The work reported in this paper was funded by the Systems Engineering for Autonomous Systems (SEAS) Defence Technology Centre established by the UK Ministry of Defence.

References

[1] P. Horn, “Autonomic computing: IBMs perspective on the state of information technology,” IBM Corporation, October 2001.

[2] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” *IEEE Computer*, vol. 36, no. 1, pp. pp. 41–50, 2003.

[3] J. Sventek, N. Badr, N. Dulay, S. Heeps, E. Lupu, and M. Sloman, “Self-managed cells and their federation,” in *Workshop Proceedings of the 17th Conference on Advanced Information Systems Engineering*. Springer-Verlag LNCS, 2005.

[4] Ponder2, “http://ponder2.net.”

[5] I. Roman-Ballesteros and C. Pfeiffer, “A framework for cooperative multi-robot surveillance tasks,” *Electronics, Robotics and Automotive Mechanics Conference, 2006*, vol. 2, pp. 163–170, Sept. 2006.

[6] D. C. MacKenzie, R. C. Arkin, and J. M. Cameron, “Multiagent mission specification and execution.” *Auton. Robots*, vol. 4, no. 1, pp. pp. 29–52, 1997.

[7] I. Rachid Alami and S. S. da Costa Bothelho, “Plan-based multi-robot cooperation,” *Advances in Plan-Based Control of Robotic Agents*, vol. Volume 2466/2002, pp. 65–95, Sept. 2002.

[8] k-team, “http://www.k-team.com.”

[9] “Policy-based management in dynamic communities,” *Annual Conference of the ITA, (ACITA)*, 2008.

[10] P. Ulam, Y. Endo, A. Wagner, and R. Arkin, “Integrated mission specification and task allocation for robot teams - design and implementation,” *Robotics and Automation, 2007 IEEE International Conference on*, pp. 4428–4435, April 2007.

[11] R. Smith, “The contract net protocol: High-level communication and control in a distributed problem solver,” *Computers, IEEE Transactions on*, vol. C-29, no. 12, 1980.

[12] L. Iocchi, D. Nardi, M. Piaggio, and A. Sgorbissa, “Distributed coordination in heterogeneous multi-robot systems,” *Autonomous Robots*, vol. 15, no. 2, pp. pp. 155–168, 2003.

[13] L. Parker, “ALLIANCE: An architecture for fault tolerant multirobot cooperation,” *Robotics and Automation, IEEE Transactions on*, vol. 14, no. 2, pp. pp. 220–240, 1998.

[14] L. Chaimowicz, V. Kumar, and M. Campos, “A paradigm for dynamic coordination of multiple robots,” *Autonomous Robots*, vol. 17, no. 1, pp. pp. 7–21, 2004.

[15] M. Likhachev, M. Kaess, Z. Kira, and R. C. Arkin, “Spatio-temporal case-based reasoning for efficient reactive robot navigation,” *Mobile Robot Labaratory, Georgia Institute of Technology*, 2005.

[16] B. P. Gerkey and M. J. Mataric, “Principled communication for dynamic multi-robot task allocation,” in *ISER ’00: Experimental Robotics VII*. London, UK: Springer-Verlag, 2001, pp. 353–362.

[17] Salemi, B. and Shen, Wei-Min, “Distributed and dynamic task reallocation in robot organizations,” *Robotics and Automation, 2002. Proceedings. ICRA ’02. IEEE International Conference on*, vol. 1, 2002.

[18] S. L. Keoh, E. Lupu, and M. Sloman, “PEACE: A policy-based establishment of ad-hoc communities,” in *ACSAC ’04: Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC’04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. pp. 386–395.