Methodologies for self-organising systems: a SPEM approach

Mariachiara Puviani
Dept. of Information Engineering, Univ. of Modena and Reggio Emilia, Modena, Italy
mariachiara.puviani@unimore.it

Giovanna Di Marzo Serugendo, Regina Frei
Computer Science and Information Systems, BBK College, London, United Kingdom
dimarzo@dcs.bbk.ac.uk, regina.frei@uninova.pt

Giacomo Cabri
Dept. of Information Engineering, Univ. of Modena and Reggio Emilia, Modena, Italy
giacomo.cabri@unimore.it

Abstract

We define 'SPEM fragments' of five methods for developing self-organising multi-agent systems. Self-organising traffic lights controllers provide an application scenario.

1. Introduction

A large amount of work has focused on translating natural self-organising mechanisms into artificial systems. These developments remain *ad hoc* solutions, usually highly dependent on finely tuned parameters. To convince potential industrial buyers of such an approach, more systematic development and validation techniques are needed. Different existing methodologies can be helpful, but not every methodology provides a solution to any problem in any context. It is therefore convenient to join reusable *method fragments* from existing methodologies. This combines the designer's need for a specific methodology with the advantages and the experience of existing and documented methodologies. The SPEM (Software Process Engineering Metamodel) specification[1] provides a complete *Meta Object Facility*[2] based metamodel, used to describe method fragments. We choose the FIPA[3] type, even though others exist. SPEM is structured as an UML profile, which facilitates the integration of different methodologies.

We report five software engineering techniques, define relevant SPEM fragments (sections 3-7) and apply them to a common self-organisation problem (section 2). Conclusions are in section 8. Abbreviations: *DL* for deliverables, *PC* for preconditions, and *GL* for guidelines.

2. Case study - Traffic Lights Control

We consider a system composed of *cars* and traffic light controllers (*TL*) disposed in a grid as shown in Figure 1. The global goal of the system is to optimise traffic throughput. Cars need to reach their destination as fast as possible without stopping; TLs need to allow vehicles to travel as fast as possible while mediating their conflicts for space and time at intersections. Traffic lights are independent of each other, and the global system behaviour will appear from the traffic flow of the whole system. To simplify the system, cars travel along horizontal and vertical lines only. TLs synchronise with each other (two by two) using the traffic flow information given by cars that pass by.

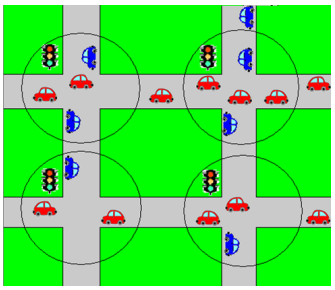

Figure 1. Traffic lights case study

3. Adelfe

ADELFE [5] is based on the RUP (Rational Unified Process) and on the AMAS (Adaptive Multi-Agent System) theory. During cooperation an agent tries to anticipate problems, detect cooperation failures *(Non Cooperative Situations - NCS)* and repair them. An AMAS agent is autonomous and unaware of the global function of the system; it can detect NCSs and acts to return to a cooperative state. Adelfe is divided into six phases or *Work Definitions* (Figure 2); each phase consists of several activities (A).

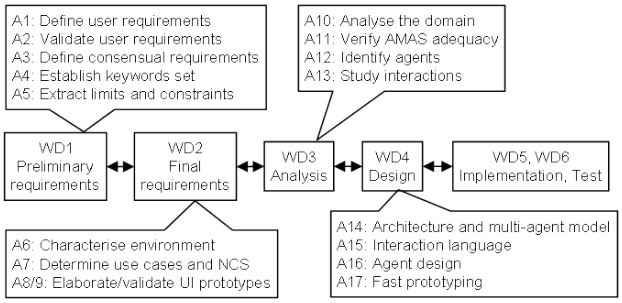

Figure 2. The ADELFE methodology [5]

1. http://www.omg.org/spec/SPEM/
2. http://www.omg.org/spec/MOF/2.0/
3. http://www.fipa.org/activities/methodology.html

Case study[4]**:** The stakeholders are TLs and cars. The AMAS adequacy is verified by answering questions of both global and local scope. Among others: 'Is the environment dynamic?' and 'Is the system linear?'. We identify two NCS: 1) when the horizontal flow is equal to the vertical one, the TL prioritises one direction; 2) when the number of cars exceeds a chosen threshold, further cars are prevented from entering the system.

SPEM: *Environmental Description Fragment* (A6 – Figure 3a). *DL:* an environment definition document and UML diagrams (scenarios) which describe the situation in the environment. *PC:* a requirement set document that defines the system requirements. *GL:* determine entities, define the context and characterize the environment.

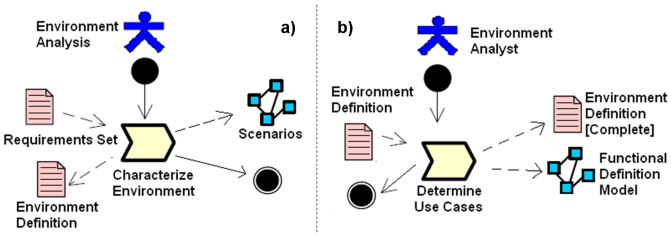

Figure 3. (a) Environment Description Fragment, (b) Use Cases Description Fragment

Use Cases Description Fragments (A7 – Figure 3b). *DL:* a functional description model, and the now completed environment definition document. *PC:* the preliminary environment definition document. *GL:* draw up an inventory of the use cases, identify cooperation failures and elaborate on sequence diagrams.

Adequacy Verification Fragment (A11 – Figure 4a). *DL:* the final AMAS adequacy synthesis document. *PC:* the preliminary software architecture document, described in the Adelfe domain description fragment. *GL:* verify the AMAS adequacy at local and global level.

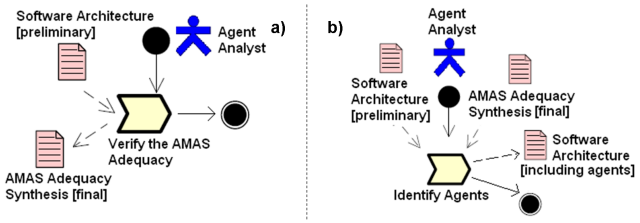

Figure 4. (a) Adequacy Verification Fragment, (b) Agent Identification Fragment

Agent Identification Fragment (A12 – Figure 4b). *DL:* the software architecture document, including the agents. *PC:* the preliminary software architecture document and the final AMAS adequacy synthesis document. *GL:* study the entities in their context, identify the potentially cooperative entities and define the agents.

4. We used the toolkit on http://www.irit.fr/ADELFE/Download.html

4. The Customised Unified Process

The Customised Unified Process (CUP) [1] is an iterative process that provides support for the design of self-organising emergent solutions in the context of an engineering process. It is based on the *Unified Process (UP)*, and is customised to explicitly focus on engineering macroscopic behaviour of self-organizing systems (see Figure 5).

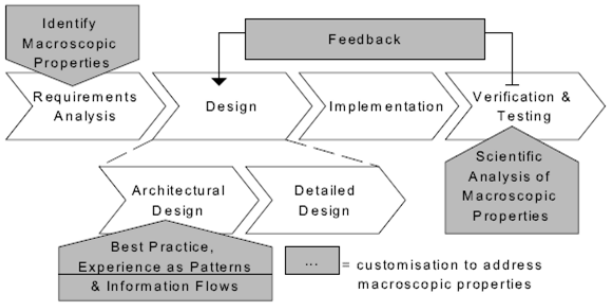

Figure 5. Customized Unified Process Methodology [1]

Requirement Analysis phase: the problem is structured into functional and non-functional requirements, using techniques such as use cases, feature lists and a domain model that reflects the problem domain. Macroscopic requirements (at the global level) are identified. *Design phase*: split into *Architectural Design* and *Detailed Design* addressing microscopic issues. *Information Flow* (a design abstraction) traverses the system and forms feedback loops. *Locality* is 'that limited part of the system for which the information located there is directly accessible to the entity' [1]. Activity diagrams are used to determine when a certain behaviour starts and what its inputs are. Information flows are enabled by decentralised coordination mechanisms, defined by provided design patterns. *Implementation phase*: the design is realised by using a specific language. When implementing, the programmer focuses on the microscopic level of the system. *Testing and Verification phase*: agent-based simulations are combined with numerical analysis algorithms for dynamical systems verification at macro-level.

Case Study: The TLs are defined as agents; cars are considered as entities. The circles around the TLs define the localities (Figure 1). The main goal can be decomposed into sub-goals, which consist of flow optimisation for each TL. The information needed for each TL is the status of its light, and the horizontal and vertical flow in its area. We chose *gradient fields* as patterns of decentralised coordination mechanism. Traffic flow automatically propagates information between TLs.

SPEM: The following fragments were extracted from the architectural design phase of the CUP.

Locality Identification Fragment (Figure 6a). *DL:* a UML diagram (e.g. an activity diagram) and a localities model. *PC:* a system requirement document that defines the system

requirements given by the users, and an agent model. *GL:* determine localities for each agent.

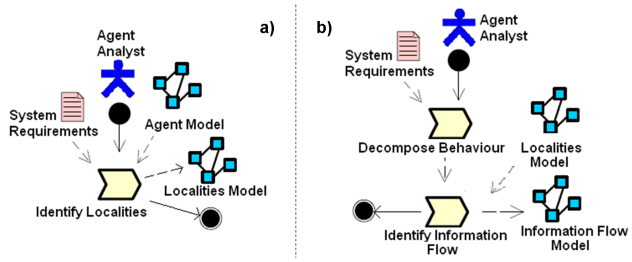

Figure 6. (a) Locality Identification Fragment, (b) Information Flow Definition Fragment

Information Flow Definition Fragment (Figure 6b). *DL:* a UML diagram (e.g. an activity diagram) and an information flow model that describes the information flow in the entire system, starting from each locality. *PC:* a system requirement document that defines the system requirements given by the users, and the localities model. *GL:* first decompose the system behaviour in sub-goals, then determine the information flow.

5. MetaSelf

MetaSelf [2] proposes both a *system architecture* and a *development process*. The MetaSelf system architecture involves loosely coupled autonomous components, repositories of metadata and executable policies, and reasoning services which dynamically enforce the policies on the basis of metadata values. Figure 7 shows the MetaSelf development method. *Requirement and Analysis phase*: identifies the functionality of the system along with self-* requirements specifying where and when self-organisation is needed or desired. *Design phase* in two steps: (a) the *patterns and mechanisms decision step*: choice of architectural patterns (e.g. autonomic manager or observer/controller architecture) and adaptation mechanism (e.g. trust, gossip, or stigmergy) (b) the *application system design step*: instantiate the chosen patterns for the specific application, architecture and policies, design the individual components (agents), select and describe the necessary metadata. *Implementation phase*: produces the run-time infrastructure.

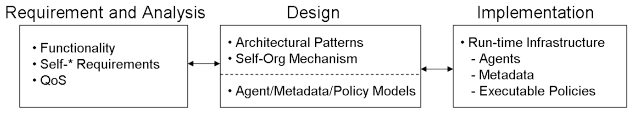

Figure 7. MetaSelf Development Process

Case Study: *Components:* The TLs are active, and the cars are passive entities (they can change their state only due to environmental change). *Self-* requirements:* Traffic flow is optimised as a result of self-organisation. *Architectural Pattern:* The generic observer/controller is chosen. Each TL comes with its observer and controller components. *Self-organization mechanism:* Gradient Fields (traffic flows) created by cars movement. Cars follow gradients, TL react to gradients by changing lights. *Coordination mechanism:* TL knows colour of light in both lanes backwards. *Policies:* e.g. Green Wave: 'Keep green light if TL backwards also has green light' or 'If horizontal flow is bigger than vertical flow (gradients fields), switch the horizontal TL to green and the vertical TL to red' and the opposite. *Metadata:* Traffic flow on each incoming lane for each TL; distance to car in each outgoing lane for each TL.

SPEM: *Identification of Pattern and Mechanism fragment* (Figure 8a). *DL:* the architectural design pattern and the adaptation and coordination mechanism. *PC:* a list of self-* requirements which represents the required system proprieties. *GL:* define the self-organisation / self-adaptation architectural design pattern and the adaptation and coordination mechanism.

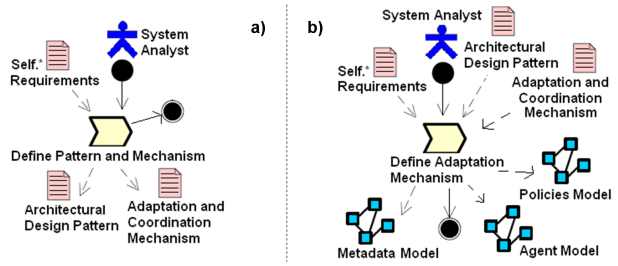

Figure 8. (a) Identification of Pattern and Mechanism Frag., (b) Identification of Software Architecture Frag.

Identification of Software Architecture Fragment (Figure 8b). *DL:* the metadata model, the agent model and the policies model. *PC:* the self-* requirements document, the architectural design patterns document and the adaptation and coordination mechanism document. *GL:* define design phase entities.

6. A General Methodology

The General Methodology [4] provides guidelines for system development in five iterative steps (Figure 9). Particular attention is given to the vocabulary used to describe self-organising systems.

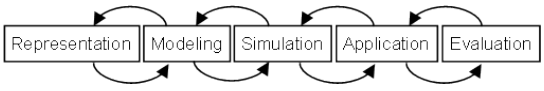

Figure 9. Methodology steps, adapted from [4]

Representation phase: the designer chooses an appropriate vocabulary, the abstractions level, granularity, variables, and interactions that have to be taken into account during system development. The representation of the system should

consider different levels of abstraction. *Modeling phase*: a control mechanism is defined, which should be internal and distributed to ensure the proper interaction between the elements of the system, and produce the desired performance. The designer should prevent the negative interferences between elements *(reduce friction)* and promote positive interferences *(promote synergy)*. *Simulation phase*: the developed model(s) are implemented and different scenarios and mediator strategies are tested. *Application phase*: develop and test model(s) in a real system. *Evaluation phase*: measure and compare performance.

Case Study: *Requirements:* develop a feasible and efficient traffic light control system. *Representation phase:* modelling on two levels: *car level* and *TL level*. Cars try to maximise their satisfaction (traveling freely and without stopping at intersections). TLs try to maximise the system's satisfaction (all cars travel as fast as possible, and are able to flow through the city without stopping). *Modeling and simulation phases:* Find a mechanism to coordinate TLs so that these mediate between cars and reduce friction. With several models iteratively refine granularity and add suitable rules. See [4] for more.

SPEM: *Control Mechanism Definition Fragment* (Figure 10). Creates a communication model based on how to optimize the system, which is relevant for self-organising systems. *DL:* a UML diagram which describes the communication protocol of the control mechanism. *PC:* an agent model and a list of constraints. *GL:* divide the labour, to promote synergies and reduce friction. The two produced documents define the model of the specified system and help during the creation of the communication and control model.

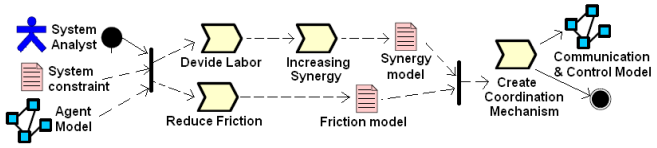

Figure 10. Control Mechanism Definition Fragment

7. A Simulation Driven Approach

The Simulation Driven Approach (SDA) to building self-organising systems [3] uses abstractions to describe the environment: the *Agent & Artefact metamodel*. *Artefacts* are passive, reactive entities providing services and functionalities to be exploited by agents through a usage interface. *Environmental agents* are responsible for managing artefacts to achieve the targeted self-* properties. Environmental agents are different form standard agents (*user agents*), which exploit artefact services to achieve individual and social goals. SDA is situated between the analysis and the design phase, as an *Early design phase*. The models are analysed using simulation, with the goal to describe the desired environmental agent behaviour and a set of working parameters (*Simulation phase*). These are calibrated in a tuning process (*Tuning phase*).

Case Study: As this approach is very similar to the one described in section 6, we do not detail it here.

SPEM: *Describe the Environment Fragment* (Figure 11). Useful in systems where the environment plays an important role, but difficult to combine with fragments from other methods which do not share this view on the environment. *DL:* UML diagrams, the agent model, the artefact model and the environmental agent model. *PC:* the system requirements. *GL:* first describe the environment by extracting agent and artefacts, then create the environmental agents which manage artefacts to achieve the system's self-* properties.

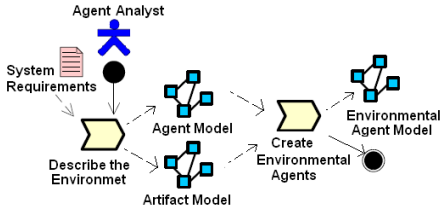

Figure 11. Environment Fragment

8. Conclusions

SPEM allows us to extract specific method fragments, to compare and reuse them. See [6] for more fragments, a second case study, a comparison of the methodologies and a fragment composition.

References

[1] T. De Wolf. *Analysing and engineering self-organising emergent applications*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, 2007.

[2] G. Di Marzo Serugendo, J. Fitzgerald, A. Romanovsky, and N. Guelfi. Metaself - a framework for designing and controlling self-adaptive and self-organising systems. Technical report, BBKCS-08-08, School of Computer Science & Information Systems, BBK College, London, UK, 2008.

[3] L. Gardelli, M. Viroli, M. Casadei, and A. Omicini. Designing self-organising environments with agents and artifacts: A simulation-driven approach. *Int. Journal of Agent-Oriented Software Engineering*, 2(2):171–195, 2008.

[4] C. Gershenson. *Design and control of self-organizing systems*. PhD thesis, Faculty of Science and Center Leo Apostel for Interdisciplinary Studies, Vrije Univ., Brussels, Belgium, 2007.

[5] G. Picard and M.-P. Gleizes. The ADELFE methodology-designing adaptive cooperative multi-agent systems. In F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 157–175. Springer US, 2004.

[6] M. Puviani, G. Di Marzo Serugendo, R. Frei, and G. Cabri. Methodologies for self-organising systems: a spem approach. Technical report, BBKCS-09-05, School of Computer Science & Information Systems, BBK College, London, UK, 2009.