

Integrating Adaptation Patterns into Agent Methodologies to Build Self-adaptive Systems

Mariachiara Puviani¹, Giacomo Cabri² and Letizia Leonardi¹

¹DIEF - Università degli Studi di Modena e Reggio Emilia, Via Vignolese 905/b, Modena, Italy

²FIM - Università degli Studi di Modena e Reggio Emilia, Via Campi, 213/A, Modena, Italy

Keywords: Multi-Agent System, Adaptation Pattern, Methodology.

Abstract: Agent systems represent a very good example of complex and self-adaptive systems. Adaptation must be conceived not only at the level of single components, but also at the system level, where adaptation must concern the entire structure of one system; adaptation patterns have been proposed to address both levels. Many methodologies have been proposed to support developers in their work, but they lack in addressing the choice and the exploitation of adaptation patterns. In this work, we propose an integration of adaptation patterns in agent-oriented methodologies, exploiting an existing methodology to concretely show how such an integration can be enacted.

1 INTRODUCTION

Nowadays intelligent software systems take places in different domains, characterised by mobility, rapid changes in operation conditions along with changes in the systems' environment, and so on. Self-adaptation has been proposed as a solution to manage the growing complexity of intelligent systems. With the continuous increasing in runtime scale and complexity of software systems, self-adaptation has assumed a central role in the software engineering development, and has often been mentioned as one of the challenges for the discipline (Fernandez-Marquez et al., 2012).

Self-adaptation is the ability of a software system or an application to automatically modify its structure and behaviour at runtime in order to ensure, maintain or recover some functional or extra-functional properties, even in the face of unexpected changes to operating conditions or user requirements.

To develop self-adaptive systems, in general two approaches exist: *parameter* adaptation and *compositional* adaptation (McKinley et al., 2004). Parameter adaptation concerns adapting the system's behaviour through changing parameters, while compositional adaptation is described as change of components (in terms of behaviour or whole structure). In our work, we focus on compositional adaptation, but in a more specific way: we do not aim to simply change the components in a system, but we aim to modify their behaviour (defined as the *pattern* that

describes it) inside the system. This will lead not to change a component, but to change its internal structure in order to make it behave differently.

In our studies, we found a *lack* of reusable and well-defined *processes* and *components* for explicitly designing and implementing self-adaptive systems. Designers often start from scratch. Moreover, the integration of the concepts into a framework that is enhanced with tools of process definition (as a well known methodology) and of means that enable to introduce adaptation, would simplify the development of self-adaptive systems and result in fast and less error-prone development.

Studying adaptation patterns, we have seen that they are considered as a useful means to introduce adaptation into a system, and more in details, in a methodology for developing self-adaptive systems.

In our work, we aim at creating a complete approach that will guide developers from the system's specification to the system's implementation. We would like to create a complete framework that permits to develop a self-adaptive system. This ongoing work is made of different steps:

1. analysis of Multi-Agent methodologies and integration of adaptation patterns into the chosen ones;
2. modification of the tools that support the chosen methodologies, for the creation of adaptive systems;
3. creation of a middleware that will merge the

concepts coming from methodologies' tools, by means of Java classes;

4. evaluation of the framework, experimenting the creation of self-adaptive systems.

As a first step, we think that integrating adaptation patterns into methodologies will aid in realising this approach of supporting the development of a self-adaptive system. To be more concrete, in this paper, we specifically apply this approach to some methodologies to develop multi-agent systems, which will represent complex and adaptive systems. In this paper, we present how adaptation patterns can be integrated in a multi-agent methodology, in order to obtain a methodology for creating self-adaptive systems.

The reminder of this paper is as follows: in Section 2, we present the Catalogue of adaptation Patterns, explaining its importance in connection with adaptive systems and methodologies. Further on, in Section 3 we introduce some selected agent-oriented methodologies, along with the criteria we used to select them, and we show how and where the Catalogue of Patterns can be included into these methodologies. In Section 5 we present some work related to our approach and at the end, in Section 6 we conclude the paper and present some future works.

2 THE CATALOGUE OF ADAPTATION PATTERNS

Closed to self-adaptation are *Service-based systems* and *Agent-based systems*. In Service-based systems, services are designed independently by different service providers and are composed to achieve a predefined goal (i.e., user tasks (Kazhamiakin et al., 2010) or business goals (Marconi et al., 2008)); while in *Agent-based systems* activities of different actors are regulated by certain collectively defined rules (Lavalin et al., 2006). In these different approaches, the focus of research on adaptation has not been put on adaptive behaviour of the whole system yet, but it is limited to the definition of adaptation solutions for specific entities (they try to adapt the behaviour of the single component and not of the whole system). Instead, our aim is to move from individual-based components to entire systems proposing adaptation techniques that support adaptation for the ensemble. To overcome this limitation we introduce the adaptation pattern approach (Puviani et al., 2013).

In literature design patterns (or simply *patterns*) are defined as reusable solutions to recurring design problems and are a mainstream of software reuse practice (Gamma et al., 1995; Morandini et al., 2010).

They crystallize a general solution to a common problem, so software developers can benefit from their reuse to develop systems. An *adaptation pattern* is a conceptual scheme that describes a specific adaptation mechanism. It specifies how the component/system architecture can express adaptivity.

An important task to develop a well performing self-adaptive system, is to understand which pattern to choose. In order to define how a pattern works in a self-adaptive system and which kind of systems is covered by a specific pattern, we wrote a Catalogue of Adaptation Patterns (Puviani et al., 2013). In this Catalogue, the different patterns are presented, and each of them describes the features of a specific adaptive system.

The adaptive behaviour inside a component or an ensemble is described in terms of feedback loops. In the Catalogue, the patterns are proposed with a specific description by means of a template, and with examples of the use of the patterns in real systems, in order to simplify the selection of the right pattern to use. The use of a pattern permits the developer to be guided to make the system exhibit a required behaviour, even when unexpected situations occur.

The use of adaptation patterns to create self-adaptive systems, has been tested in different field, in many applications (Mayer et al., 2013; Puviani et al., 2014b), and guarantees correct results in systems that are frequently changing, not only in their internal conditions, but also in the environment where they are operating.

As said in the Introduction, the use of a methodology could be very useful to develop self-adaptive systems. However, the current methodologies consider adaptation only at level of single components, instead of at the system's level. That is the reason why we consider necessary to introduce the Catalogue of Patterns into methodologies: it will enact adaptivity at the level both of single component, and of the entire system. In fact the Catalogue of Patterns will support the methodologies in the creation of an adaptive system where the structural adaptation of the whole system is considered very relevant.

3 AGENT-ORIENTED METHODOLOGIES FOR ADAPTIVE SYSTEMS

In order to support application developers during the creation of an adaptive system, it is necessary to provide a methodology that support adaptation mechanisms starting from the system requirements. The ini-

tial idea is not to propose a methodology from scratch, but to have as a base a stable and well known methodology.

Moreover, we consider that “agents” are one of the most useful paradigms to build intelligence distributed systems, so we would like to use that paradigm to create adaptive systems. To do that, we started from the study of agent-oriented methodologies as a starting point to introduce adaptation features while building a system.

Considering MASs (Multi Agent Systems), it is generally accepted that analysis and design of agent-based systems require an Agent-Oriented Software Engineering (AOSE) methodology. There are now many mature AOSE methodologies (Henderson-Sellers and Giorgini, 2005), (Bergenti et al., 2004), including MaSE (DeLoach et al., 2001), Tropos (Bresciani et al., 2004), Gaia (Zambonelli et al., 2003), Prometheus (Padgham and Winikoff, 2005), INGENIAS (Pavón et al., 2005), ADEM¹, ADELFE (Bernon et al., 2003), SODA (alice Research Group et al., 2009) and PASSI (Cossentino, 2005).

After different studies (Puviani et al., 2012), we found out that to create a unified methodology that may have the most powerful features of every of the starting ones, is very difficult. For example, we are not able to prove if a new unified methodology covers all the possible scenarios, as happened for MAR&A (Cabri et al., 2009), that is a composed methodology, but is not applicable to adaptive systems. Moreover, not all the composing methodologies use the same language or the same concepts, and translating them into unified terms will not be always easy. Furthermore, creating a new methodology for adaptive systems from scratch will not be easy as well. It may be yet another methodology, and there is no guarantee that it will be able to build all the adaptive systems.

All these reasons suggested us to not create a methodology dedicated to self-adaptive system, or to compose a methodology, but to start from well known and well defined methodologies, and to insert the Catalogue of Patterns into them in order to have self-adaptive feature.

Starting from our previous work (Puviani et al., 2010), we found out that some AOSE methodologies, even if they are well known, are not up to date, or no more utilised to develop intelligent complex agents systems. So we selected only few methodologies that we consider suitable to build a self-adaptive system. The methodologies that we selected have some common features:

- they are updated (e.g. a new version of the methodology has been released in the last years);
- they have been tested in different distributed systems;
- they use well known paradigms like UML or the SPEM approach (Seidita et al., 2009), (Puviani et al., 2009) that will be very useful in order to introduce adaptation patterns;
- they use the concept of “role” to define adaptation patterns in a system;
- they have a supporting tool, or specific indication for the development of a system.

The methodologies we selected for our work are: ADELFE, PASSI2 and SODA. For space reasons, in this paper we describe only PASSI2, along with the indications on where introducing the Catalogue of Patterns.

As said before, a common point of these methodologies is that all of them have been described using the SPEM (Software Process Engineering Metamodel) approach (Object Management Group, 1997). This will be useful in order to insert the Catalogue of Patterns in terms of SPEM fragments. In this way, it will be possible to better define the concepts presented in patterns and to insert them into the different methodologies.

To improve reading of the paper, in Figure 1 we report the definitions of some common notations used by SPEM.









WorkProduct		Anything produced, consumed, or modified by a process.
WorkDefinition		Operation that describes the work performed in the process.
Activity		The main subclass of WorkDefinition, it describes a piece of work performed by one ProcessRole.
ProcessRole		Defines a performer for a set of WorkDefinitions in a process. It represents abstractly the "whoole process" or one of its components.
ProcessPackage		
Phase		A specialization of WorkDefinition. Its precondition defines the phase entry criteria and its goal defines the phase exit criteria.
Document		A WorkProduct
UMLModel		A WorkProduct

Figure 1: SPEM notations.

¹<http://www.whitestein.com/adem>

4 EXAMPLE: PASSI2

PASSI2 (Process for Agent Society Specification and Implementation) (Cossentino and Seidita, 2009) is the evolution of PASSI (Cossentino, 2005), a methodology that aims at covering all the phases of a system development from the requirements' analysis to the deployment configuration, coding, and testing.

It is based on a meta-model describing the elements that constitute the system to be designed (agents, tasks, communications, roles) and what are the relationships among them. The importance of this description is in the lack of a universally accepted meta-model of MASs (differently from object-oriented systems) that makes unclear any agent design process that does not precisely define the structure of the system it aims to produce. PASSI2 has been designed keeping in mind the possibility of designing systems with the following peculiarities: (i) highly distributed, (ii) subject to a (relatively) low rate of requirements changes, (iii) openness (at runtime external systems and agents that are unknown at design time will interact with the system to be built). Robotics, workflow management, and information systems are the specific application areas where it has been widely applied.

PASSI2 is composed of three models that address different design concerns and nineteen phases, as we can see in Figure 2. An important aspect of PASSI2 is that it uses standards as UML and adapts it to the need of representing agent systems through its extension mechanisms (constraints, tagged values and stereotypes).

Synthetically, the models and phases of PASSI2 are:

1. **System Requirements Model.** A model of the system requirements in terms of agency and purpose. Developing this model involves:
 - **Domain Description (DD).** A functional description of the system using conventional use-case diagrams.
 - **Agent Identification (AId).** Separation of responsibility concerns into agents, represented as UML packages.
 - **Role Identification (RIId).** Use of sequence diagrams to represent each agent's responsibilities through role-specific scenarios.
 - **Agent Structure Exploration (ASE).** An analysis-level description of the agent structure in terms of tasks required for accomplishing the agent's functionalities.
 - **Task Specification (TSp).** Specification through state/activity diagrams of the capabilities of each agent.
2. **Agent Society Model.** A model of the social interactions and dependencies among the agents involved in the solution. Developing this model involves five phases:
 - **Domain Ontology Description (DOD).** Use of class diagrams to describe domain categories (concepts), actions that could affect their state and propositions about values of categories.
 - **Communication Ontology Description (COD).** Use of class diagrams to describe agents' communications in terms of referred ontology, interaction protocol and message content language.
 - **Role Description (RD).** Use of class diagrams to show distinct roles played by agents, the tasks involved what the roles involve, communication capabilities and inter-agent dependencies in terms of services.
 - **Multi-Agent Structure Definition (MASD).** Use of conventional class diagrams to describe the structure of solution agent classes at the social level of abstraction.
 - **Multi-Agent Behaviour Description (MABD).** Use of activity diagrams or state-charts to describe the behaviour of individual agents at the social level of abstraction.
3. **Implementation Model.** A model of the solution architecture in terms of classes, methods, deployment configuration, code and testing directives; it is composed of seven phases, the first two are performed at both the multi-agent (whole agent society) and single-agent abstraction level:
 - **Single-Agent Structure Definition (SASD).** Use of conventional class diagrams to describe the structure of solution agent classes at the implementation level of abstraction.
 - **Single-Agent Behaviour Description (SABD).** Use of activity diagrams or state-charts to describe the behaviour of individual agents at the implementation level of abstraction.
 - **Deployment Configuration (DC).** Use of deployment diagrams to describe the allocation of agents to the available processing units and any constraints on migration, mobility and configuration of hosts and agent-running platforms.
 - **Code Reuse (CR).** A library of patterns with associated reusable code to allow the automatic generation of significant portions of code.
 - **Code Completion (CC).** Source code of the target system that is manually completed.
 - **Agent Test.** Verification of the single behaviour with regards to the original requirements of the system solved by the specific agent.

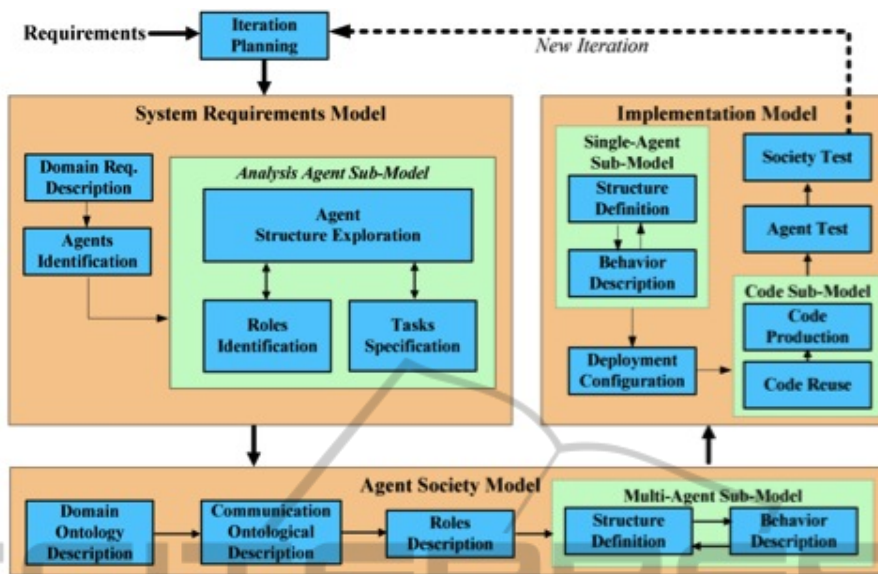


Figure 2: PASSI2 models and phases.

- **Society Test.** Validation of the correct interaction of the agents, performed in order to verify that they actually concur in solving problems that need cooperation. This test is done in the most real situation that can be simulated in the development environment.

The Iteration Planning activity is positioned at a higher level of abstraction, above the logical sequence of models and phases. It is at the base of every iterative incremental process and in our case consists of the analysis of the Problem Statement and all the other available documents (for instance outputs of previous iterations) in order to identify the requirements (and related risks) that should be faced in the next iteration (that is considered as the nineteen phase).

An important concept in PASSI2 is that of “role”. A *role* is defined by the set of responsibilities defining the subjective behaviour of an agent in an interaction (conversation) with another one or in providing some service in one or more scenarios; an agent may play one or more roles at the same time. Roles are very important because they are considered a useful paradigm that can be used to define the different patterns in a system (Puviani et al., 2014a).

Two are the main phases that involved roles: the *Role Identification phase*, into the System Requirements Model (Figure 3), and the *Role Description phase* into the Agent Society Model (Figure 4).

The *Roles Identification* phase produces a set of sequence diagrams that specify scenarios from the agents’ identification use case diagram. In this phase, the Catalogue of Patterns is added as input, in order to create specific roles able to describe an adap-

tive system. In this context, it is also particularly important to investigate all the paths involving inter-agent communications, and fortunately some guidelines can be considered: (1) such communication paths are shown in the AId diagram by the presence of a relationship between two agents with the communication/instantiation stereotype; (2) each relationship may belong to several scenarios; (3) for each relationship in a specific scenario of the AId diagram, there is at least one message in the sequence diagram of the RId phase. In that phase, roles are identified in the sense that agents’ external manifestations are captured in sequence diagrams where agents participate playing one or more roles concurring to the evolution of the system dynamic.

Moreover, the Catalogue of patterns is introduced in the *Roles Description* phase. This phase consists in modelling the lifecycle of each agent, looking at the roles it can play, the collaboration it needs, the communications in which it participates and, with the inclusion of the Catalogue of Patterns, the adaptive system to develop. In the RD diagram all the rules of the society, laws of the society and the domain in which the agent operates are introduced. They could be expressed in plain text or OCL² in order to have a more precise, formal description.

In PASSI2, a role is a portion of the social behaviour of an agent that is characterized by some specifications such as a set of attributes (for example responsibilities, permissions, activities, and protocols) or providing a functionality/service. Most commonly, roles are devoted to provide services, share

²Object Constraint Language

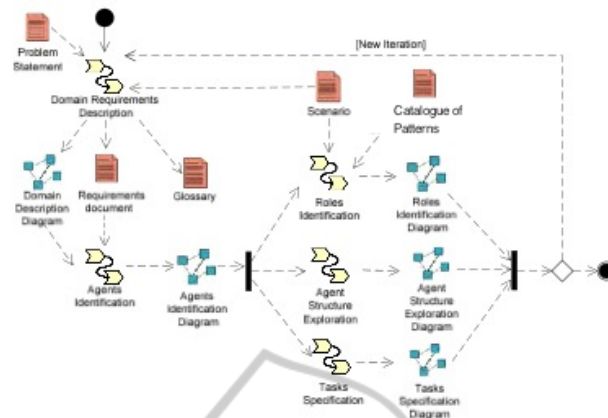


Figure 3: PASSI2: System Requirements Model activities and resulting work products.

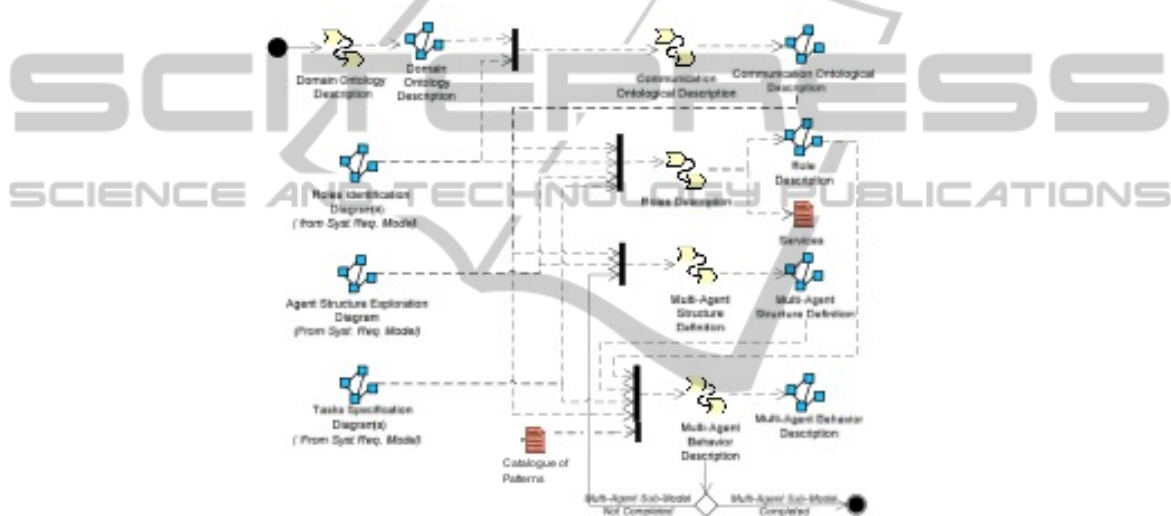


Figure 4: PASSI2: Agent Society Model activities and resulting work products.

resources or achieving a goal (this is always related to ensuring the fulfilment of the functionalities that can be deduced from the use cases assigned to the agent). The defined RD diagram is a class diagram where roles are classes grouped in packages representing agents. Roles can be connected by relationships representing changes of role, by dependencies for a service or the availability of a resource and by communications.

Specifically, in the *Agent Society Model*, the Catalogue of Patterns is introduced for the Multi-Agent Behaviour Description (MABD), where agents are described in terms of their behaviour both from the social-exterior point of view and the internal flow of control, as we can see in Figure 4. Here the Catalogue is necessary to identify which role to choose to obtain the system adaptation in the considered environment.

PASSI2 does not have a real Code Production Phase, but each programmer has to complete the code

of the application starting from the design of the skeletons produced by the methodology.

The PASSI2 design methodology is supported by a specific design tool, granting a large number of automatisms during the design, and a pattern repository for the reuse practice; these are determinant in cutting down the time and cost for developing systems (Chella et al., 2004). The toolkit is PTK (PASSI ToolKit). The PTK add-in can generate the code for all the skeletons of the agents, tasks and other classes included in the project. The pattern repository consists of a serie of reusable portions of agents and tasks. The repository also includes a list of tasks that can be applied to existing agents.

5 RELATED WORK

In literature, many approaches on patterns for self-

adaptation exist, like the one of (Ramirez and Cheng, 2010), (Cabri et al., 2011) and (Weyns et al., 2012). However, in this paper we do not focus on the use of adaptation pattern to create self-adaptive systems, but on the definition of a useful methodology to create this kind of systems, with the aid of the patterns' approach.

In the last years, engineering research has tackled a well-defined problem and has carefully selected and combined existing solution into a comprehensive development framework for self-adaptive systems.

A lot of projects like MADAM project³ and the MUSIC project⁴ tried to address adaptation in different scenarios, from both the theoretical and the practical perspective. For example, the MUSIC project (Hallsteinsen et al., 2012) would like to introduce a model-driven development methodology (Geihs et al., 2009) for self-adaptive context-aware applications. Different from us, this approach was to write a new methodology instead of exploiting the power of existing ones.

Other approaches as CARISMA (Capra et al., 2003) and RAINBOW (Garlan et al., 2004) propose self-adaptation middleware or architectural styles to develop self-adaptive software, but they do not propose any methodology that will guide developers from the collection of requirements to implementation. Moreover, MOCAS (Model of Components for Adaptive Systems) propose a generic state-based component model which enables the self-adaptation of software components along with their coordination (Ballagny et al., 2009); but like the other approaches, there are not concrete guidelines, considered as a methodology.

6 CONCLUSIONS

In this paper, we propose an approach to enrich methodologies for addressing adaptation in building self-adaptive systems. We considered in particular some specific agent-oriented methodologies, and we introduced in these the Catalogue of Patterns that will help in defining self-adaptive systems. We think that this enrichment can be done in any methodologies for building adaptive systems. The possibility of easily inserting the Catalogue of Patterns inside a method-

³Mobility and Adaptation-enabling Middleware, supported by the European Union under research grant 004159 lasting from September 2004 to March 2007.

⁴Self-Adapting Applications for Mobile Users in Ubiquitous Computing Environments, supported by the European Union under research grant IST-035166 lasting from October 2006 to March 2010.

ology (using the SPEM notation), permits harnessing the power of the selected methodologies.

As an ongoing work we are testing these modified methodologies in different scenarios, to have quantitative and qualitative results of the effectiveness of the methodologies. Moreover, as future work, we would like to complete the methodologies process, introducing the Catalogue of Patterns also in the supporting tools, in order to have all the steps completed. Then we will create a framework that will permits matching the methodologies' concepts into agents' infrastructures.

ACKNOWLEDGEMENTS

The work is supported by the "Linea strategica SMART ICT FOR SMART SOCIAL WORLDS" of the Università di Modena e Reggio Emilia.

REFERENCES

- aliCE Research Group et al. (2009). Soda home page.
- Ballagny, C., Hameurlain, N., and Barbier, F. (2009). Mocas: A state-based component model for self-adaptation. In *Self-Adaptive and Self-Organizing Systems, 2009. SASO'09. Third IEEE International Conference on*, pages 206–215. IEEE.
- Bergenti, F., Gleizes, M.-P., and Zambonelli, F. (2004). *Methodologies and software engineering for agent systems: the agent-oriented software engineering handbook*, volume 11. Springer.
- Bernon, C., Gleizes, M.-P., Peyruqueou, S., and Picard, G. (2003). Adelfe: A methodology for adaptive multi-agent systems engineering. In *Engineering Societies in the Agents World III*, pages 156–169. Springer.
- Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236.
- Cabri, G., Puviani, M., and Leonardi, L. (2009). The mar&a methodology to develop agent systems. In *ICAART*, pages 501–506.
- Cabri, G., Puviani, M., and Zambonelli, F. (2011). Towards a taxonomy of adaptive agent-based collaboration patterns for autonomic service ensembles. In *CTS*, pages 508–515. IEEE.
- Capra, L., Emmerich, W., and Mascolo, C. (2003). Carisma: Context-aware reflective middleware system for mobile applications. *Software Engineering, IEEE Transactions on*, 29(10):929–945.
- Chella, A., Cossentino, M., and Sabatucci, L. (2004). Tools and patterns in designing multi-agent systems with passi. *WSEAS Transactions on Communications*, 3(1):352–358.

- Cossentino, M. (2005). From requirements to code with the passi methodology. *Agent-oriented methodologies*, 3690:79–106.
- Cossentino, M. and Seidita, V. (2009). Passi2—going towards maturity of the passi process.
- DeLoach, S. A., Wood, M. F., and Sparkman, C. H. (2001). Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(03):231–258.
- Fernandez-Marquez, J. L., Serugendo, G. D. M., Snyder, P. L., and Valetto, G. (2012). A pattern-based architectural style for self-organizing software systems. *Drexel University, Department of Computer Science, Tech. Rep.*, 6.
- Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns*. Addison Wesley, Reading (MA).
- Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., and Steenkiste, P. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54.
- Geihs, K., Reichle, R., Wagner, M., and Khan, M. U. (2009). Modeling of context-aware self-adaptive applications in ubiquitous and service-oriented environments. In *Software engineering for self-adaptive systems*, pages 146–163. Springer.
- Hallsteinsen, S., Geihs, K., Paspallis, N., Eliassen, F., Horn, G., Lorenzo, J., Mamelli, A., and Papadopoulos, G. A. (2012). A development framework and methodology for self-adapting applications in ubiquitous computing environments. *Journal of Systems and Software*, 85(12):2840–2859.
- Henderson-Sellers, B. and Giorgini, P. (2005). *Agent-oriented methodologies*. IGI Global.
- Kazhamiakin, R., Paolucci, M., Pistore, M., and Raik, H. (2010). Modelling and automated composition of user-centric services. In *On the Move to Meaningful Internet Systems: OTM 2010*, pages 291–308. Springer.
- Lavinal, E., Desprats, T., and Raynaud, Y. (2006). A generic multi-agent conceptual framework towards self-management. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 394–403. IEEE.
- Marconi, A., Pistore, M., and Traverso, P. (2008). Automated composition of web services: the astro approach. *IEEE Data Eng. Bull.*, 31(3):23–26.
- Mayer, P., Klarl, A., Hennicker, R., Puviani, M., Tiezzi, F., Pugliese, R., Keznikl, J., and Bure, T. (2013). The autonomic cloud: a vision of voluntary, peer-2-peer cloud computing. In *Self-Adaptation and Self-Organizing Systems Workshops (SASOW), 2013 IEEE 7th International Conference on*, pages 89–94. IEEE.
- McKinley, P. K., Sadjadi, S. M., Kasten, E. P., and Cheng, B. H. (2004). Composing adaptive software. *Computer*, 37(7):56–64.
- Morandini, M. et al. (2010). On the use of the Goal-Oriented Paradigm for System Design and Law Compliance Reasoning. In *iStar*, volume 586, pages 71–75. CEUR-WS.org.
- Object Management Group (1997). SPEM. <http://www.omg.org/technology/documents/formal/spem.htm>.
- Padgham, L. and Winikoff, M. (2005). *Developing intelligent agent systems: A practical guide*, volume 13. John Wiley & Sons.
- Pavón, J., Gómez-Sanz, J. J., and Fuentes, R. (2005). The ingenias methodology and tools. *Agent-oriented methodologies*, 9:236–276.
- Puviani, M., Cabri, G., and Leonardi, L. (2014a). Enabling self-expression: the use of roles to dynamically change adaptation patterns. In *FOCAS 2014*. IEEE Computer Society.
- Puviani, M., Cabri, G., and Zambonelli, F. (2013). A taxonomy of architectural patterns for self-adaptive systems. In *Proceedings of the International C* Conference on Computer Science and Software Engineering*, pages 77–85. ACM.
- Puviani, M., Cabri, G., and Zambonelli, F. (2014b). Agent-based simulations of patterns for self-adaptive systems. In *ICAART 2014 - Proceedings of the 6th International Conference on Agents and Artificial Intelligence, Volume 1, ESEO, Angers, Loire Valley, France, 6-8 March, 2014*, pages 190–200. IEEE Computer Society.
- Puviani, M., Cossentino, M., Cabri, G., and Molesini, A. (2010). Building an agent methodology from fragments: the mensa experience. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 920–927. ACM.
- Puviani, M., Serugendo, G. D. M., Frei, R., and Cabri, G. (2009). Methodologies for self-organising systems: a spem approach. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 02*, pages 66–69. IEEE Computer Society.
- Puviani, M., Serugendo, G. D. M., Frei, R., and Cabri, G. (2012). A method fragments approach to methodologies for engineering self-organizing systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 7(3):33.
- Ramirez, A. and Cheng, B. (2010). Design patterns for developing dynamically adaptive systems. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 49–58. ACM.
- Seidita, V., Cossentino, M., and Gaglio, S. (2009). Using and extending the spem specifications to represent agent oriented methodologies. In *Agent-Oriented Software Engineering IX*, pages 46–59. Springer.
- Weyns, D., Malek, S., Andersson, J., and Schmerl, B. (2012). Introduction to the special issue on state of the art in engineering self-adaptive systems. *Journal of Systems and Software*, 85(12):2675–2677.
- Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003). Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370.