A Taxonomy of Architectural Patterns for Self-Adaptive Systems

Mariachiara Puviani, Giacomo Cabri and Franco Zambonelli
Università degli Studi di Modena e Reggio Emilia, Italy
name.surname@unimore.it

ABSTRACT

Autonomic systems are able to adapt themselves to unpredicted and unexpected situations. Such adaptation capabilities can reside in individual *components* as well as in *ensembles* of components. In particular, a variety of different architectural patterns can be conceived to support self-adaptation at the level both of components and of ensembles. In this paper, we propose a classification of such self-adaptation patterns – for both the *component* level and the *system* level – by means of a taxonomy organized around the locus in which the feedback loops promoting adaptation reside. We show that the proposed classification covers most self-adaptation patterns, and enables deriving further ones by applying a simple set of composition mechanisms. Three examples of existing patterns of the taxonomy are detailed in the paper to show the applicability of the approach. As discussed in the paper, the advantage of the proposed classification is twofold: it enables identifying the (possibly common) properties of the existing self-adaptation patterns; and, consequently, it can help developers in choosing the most appropriate self-adaptation patterns for the development of autonomic systems.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications

General Terms

Theory

Keywords

Taxonomy, adaptativity, pattern

1. INTRODUCTION

Autonomic systems are characterized by being able to autonomously adapt (i.e., *self*-adapt) their behaviour and structure in response to dynamically changing situations and contingencies, so as to carry out their task with satisfactory quality despite unexpected execution conditions [13]. The capability of self-adaptation is typically obtained by means of one ore more feedback loops integrated in the system itself. Such feedback loops enable the system and/or its components to dynamically "sense" what is happening in the operational environment, so as to plan and enact corrective actions at the level of system *configuration*, *behaviour*, and/or *structure* [21].

The need for having feedback loops in a system raises the issue of determining how such feedback loops can be integrated in it, i.e., according to which architectural scheme. The seminal Autonomic Computing Reference Architecture (ACRA) [14] suggests the concept of *autonomic managers* (AM) as the entities that close a feedback loop on a system. However, a number of different architectural solutions can be devised for how to integrate multiple, possibly interacting, autonomic managers in a system [5], to close multiple feedback loops around components and/or ensembles. Moreover, solutions can be thought in which feedback loops are implicitly embedded in the interactions between components and/or their operational environment [6]. Even if we start our study from the ACRA model, we aim at extending it to make our classification complete.

The process of engineering complex autonomic systems could definitely take advantage of studies detailing the many possible *architectural patterns for self-adaptation via feedback loops* (from now on simply "self-adaptation patterns"). Indeed, some early studies that analyse the possible self-adaptation patterns exist [20, 6, 23]. What is still missing is a comprehensive analysis of the different self-adaptation patterns, possibly integrated by their rational classification and organization.

Besides its conceptual importance, a comprehensive classification of self-adaptation patterns, associated with a description of each pattern, can be useful for two reasons: *(i)* it can enable identifying common properties in the existing self-adaptation patterns, relations between patterns, and the mechanisms to switch from one pattern to another (as a sort of meta-level structural adaptation [21, 25]); *(ii)* it can help developers in choosing the most appropriate self-adaptation pattern for the requirements of their systems, pointing out not only the features of individual patterns but also facilitating the analysis of possible alternate choices [9].

Along the above lines, the contribution of this paper is to propose a strategy for the *classification* of the self-adaptation patterns, leading to a grid-based two-dimensional *taxonomy*. The strategy is centred around the analysis of the *loci* in

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
C^3S^2E-13 2013, July 10-12, Porto [Portugal]
Copyright 2013 ACM 978-1-4503-1976-8/12/06 ...$15.00.

which feedback loops are integrated — at the level of *individual components* or at the level of *ensemble*. We show that, by applying this strategy, a large number of self-adaptation patterns can be described and comparatively analysed, and the mechanisms to compose more complex patterns can be easily identified. Accordingly, the resulting *catalogue of self-adaptation patterns* can result in a more useful tool than simply an unordered list of different architecture schemes.

The remainder of this paper is organized as follows. Section 2 discusses related work in the area. Section 3 introduces the criteria underlying our classification of self-adaptation patterns and draws the resulting taxonomy. Section 4 presents some examples of pattern in the taxonomy, as part of a complete catalogue, and also discusses how to compose more complex patterns. Section 5 concludes the paper and sketches future research directions.

2. RELATED WORK

The interest in engineering self-adaptive systems is growing, as shown by the number of recent surveys and overviews on the topic [7, 21]. However, a comprehensive and rationally-organized analysis of architectural patterns for self-adaptation is still missing, despite the potential advantages (outlined in the introduction) of such a contribution.

Salehie and Tahvildari [21] survey and classify the various principles underlying self-adaptation and the means by which adaptation can be enforced in a system, i.e., the different mechanisms to promote adaptation at the behavioural and structural level. Similarly, Andersson et al. [2] propose a classification of modelling dimensions for self-adaptive systems to provide the engineers with a common set of vocabulary for specifying the self-adaptation properties under consideration and select suitable solutions. However, and although both these works emphasize the importance of feedback loops, nothing is said about the patterns by which such feedback loops can be organized to promote self-adaptation.

Gomaa et al. [11] focus on the mechanisms to perform adaptation actions, and on the various schemes that should be adopted to perform such adaptation actions at run-time and in a safe way. Although they adopt the term "adaptation patterns", and as important as their contribution could be to support run-time adaptation, they say nothing on the architectural patterns for the feedback loops that can identify and enact adaptation actions. Ramirez and Cheng [20] go somehow farther, by expanding upon the most common patterns for adaptation actions and representing them – the same as we do – in a more standard "pattern template". Again, though, the focus is different from that of a detailed and comprehensive analysis of architectural self-adaptation patterns.

Coming to work that have a more direct relation with ours, Brun et al. [5] present a possible classification of self-adaptive systems with the emphasis on the use of feedback loops as first-class entities in control engineering. They unfold the role of feedback loops as a general mechanism for self-adaptation, essential for understanding all types of self-adaptation. Taking inspiration for control engineering, natural systems and software engineering, the authors present some self-adaptive architectures that exhibit feedback loops. They also identify the critical challenges that must be addressed to enable systematic and well-organized engineering of self-adaptive and self-managing software systems. Their analysis of different kinds of feedback loops is very relevant for our work, and in our effort towards a comprehensive and complete taxonomy of patterns for feedback loops we have partially built upon it.

Grounded on earlier works on architectural self-adaptation approaches, the FORMS model [22] (FOrmal Reference Model for Self-adaptation) enables engineers to describe, study and evaluate alternative design choices for self-adaptive systems. FORMS defines a shared vocabulary of adaptive primitives that – while simple and concise – can be used to precisely define arbitrary complex self-adaptive systems, and can support engineers in expressing their design choices, there included those related to the architectural patterns for feedback loops. FORMS does not have the ambition to analyse and classify architectural self-adaptation patterns, and rather has to be considered as a potentially useful complement to our work.

Closest to our approach, Weyns et al. [23] introduce the concept of patterns for self-adaptive systems based on control loops. The paper describes how control loops are able to enforce adaptivity in a system and presents a set of patterns. Differently from our scope this paper does not aim to describe a comprehensive set of patterns for adaptive systems, but basically identifies in the patterns, how different MAPE loops interact with each other.

In summary, the analysis of the related work in the area shows that: *(i)* there is the need for analysing the various patterns that can be adopted to promote and support self-adaptation; *(ii)* nevertheless, a comprehensive taxonomy of architectural self-adaptation patterns – as the one we propose in this paper – is missing.

3. THE TAXONOMY OF PATTERNS

In this section we introduce the basic models and mechanisms that form the self-adaptation patterns of our catalogue and that enable organizing a taxonomy of such pattern. In particular:

- we introduce our basic model of components;
- we detail the different types of components we accommodate in our schemes;
- we describe how these components can be composed to create self-adaptive patterns at the component level and at the ensemble level;
- we present the complete taxonomy of self-adaptive patterns.

3.1 Model of Basic Components

Our work is situated in the context of the ASCENS project (www.ascens-ist.eu) [15], which approaches the development of large-scale self-aware autonomic service systems with a formally grounded component-based approach.

The basic component is thus a context-aware and controllable *service components* (SC). In particular:

- SCs provide an interface to communicate with other components and provide/request services. However, components may also have internal, proactive rather than reactive, threads of execution.
- SCs are situated, that is they live an act in a computational or physical environment, can sense the property of such environment, decide to adapt its behaviour to it, and possibly act in the environment.

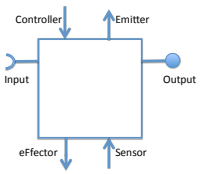

Figure 1: The structure of service components

- SCs tolerate being inspected by other components (autonomic managers, typically), and being modified in their internal working parameters.

A clear definition of the components' interfaces helps in understanding the mechanism of components composition [6]. The interfaces of a SC (see Figure 1) can be described as a tuple $\langle I, O, S, F, E, C \rangle$.

- *Input*, used to receive information (e.g. service's request);
- *Output*, used to send information (e.g. service's reply);
- *Sensor*, that makes the component able to achieve information from the external (e.g. others components and/or the environment);
- *eFfector*, that makes the component be able to manage the external (e.g. act on the environment, manage other components);
- *Emitter*, used to emit status information to an external manager. This interface permits also to share information taken from the environment (using sensors) or other components;
- *Controller*, that makes it possible to an external adaptation manager to change and adapt the component's internal state.

As said before, the Emitter and the Controller are introduced to enable autonomicity in the SC via the possibility to connect an external AM. Sensors and eFfectors are both used to make a component communicate with the environment or with another component (that extracts/gives information using Controllers and Emitters).

3.2 Types of Basic Components

A component is defined as adaptive when its decisions and the interactions outcome are defined at run time, on the base of the current status of the entire system (environment, resources, interactions with other components, etc).

As we said before, feedback loops enable to describe how adaptation occurs, as it is widely recognized in the area of self-adaptive systems (e.g. [5, 7, 23]). These feedback loops can be internal or external to the component and this reflects an internal or external adaptation. This also implies that, somehow, there exist means to inspect and analyse what is happening in the system and to have the components react accordingly. So, in the proposed taxonomy, we classify adaptive patterns that describe SCs in terms of how feedback loops are enacted.

Starting from the model of components, we defined in our taxonomy three types of basic components, described in the following and in a more detailed way in [17]. These components are considered the basic ones because we can explain them in term of feedback loops: *absent*, *internal* to the component, *external* to the component. These basic components, which show different extents of adaptivity, can be composed in different ways to create a number of self-adaptation patterns, either at the level of individual components or at the level of ensemble.

The "Reactive Service Component" (SC-Reactive) presents a SC that is able to react to service's requests. The component receives services' requests as Input and replies to them. Inside the component there are some "actions" ($A1$, $A2$, ..., An) that can be performed by the component, and a "logic" that selects the most appropriate action in order to satisfy a service's request. The component is also situated in an environment and it senses what is happening in the environment (and this influences its actions) and can also act on the environment. We emphasize that the component's reaction is mediated by its logic: this means that when a request $R1$ arrives, the component will initially respond with action $A1$; but if changes in the environment occur, the logic can choose another action (e.g. $A2$ instead of $A1$) that better responds to the same request $R1$. This different action is chosen in order to better satisfy the request, due to the adaptation of the component in reaction to external changes. In this component there is no feedback loop, so its adaptivity is a limited one because it is a simple reaction to the external environment. An example of a SC-Reactive is a simple robot, acting as a sort of bee that is part of a swarm that randomly explores the environment. This reactive component walks around, but does nothing else without an external stimulus; when it receives a stimulus, it reacts accordingly. The only coordination occurring between robots is in the form of simple exchange of local signals (e.g., virtual pheromones). It has no knowledge about the external environment and about the other components of the swarm. A robot, while exploring, deposits pheromone trials, and its random movements have a preferred direction that tends to move it away from existing pheromone trials.

The "Proactive Service Component" (SC-Proactive) represents a *goal-oriented* component that has a feedback loop inside. This component has a higher degree of autonomy, and thus of self-adaptation, than the Reactive one; but the adaptivity is embedded into the component and it is not controllable. It can code internal goals that, along with the logic, can manage the choice of a specific action in response to the service request. The choice can depend on: (i) how close to the goal each action will bring the component, possibly also based on the effects of past actions, and (ii) the current status of the environment in which it is situated. The internal structure for action selection based on goals, current status, and environment, makes the component structured as implicitly having an internal autonomic feedback loop. Nevertheless this component shows the same composability property as a SC-Reactive. An example of a SC-Proactive is a goal-oriented robot, embedding in its goal-oriented activities, the capability of perceiving the status of the environment and of adaptively selecting the best plan of actions. For example the robot, while walking in the environment, has to search for objects and carry them to a specific place. The component is at the same time *reactive*, *proactive* and *social* as defined in [24]. The main difference between this robot and the reactive one is that this robot is

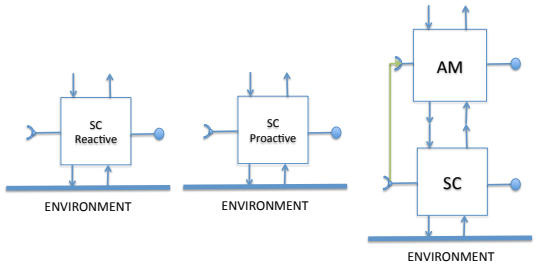

Figure 2: Basic types for individual components: reactive (left), proactive (center) and autonomic manager - and shown already associated to a SC (right)

able to take the initiative in order to transport an object, and to select the most suitable plan to do that. Because of its sociality, robots exchange information about the explored environment and coordinate with each other to avoid searching for objects in areas already explored.

The "Autonomic Manager Service Component" (AM) externalizes and makes explicit a feedback loop. The AM does not need to be a complex component. It has an interface similar to the one of the SC, but its internal logic is different because it has a different role. For example, to adapt a SC, the AM can simply set a flag that will change the logic of the SC in terms of which action to use to respond to a specific request. Or else, the AM can act changing the set of actions of the SC. The Input of the AM is the same of the component it manages, to make its monitoring of the component complete. In addition, in order to manage the component, the AM uses its own Sensors to extract the information about the internal state of the component from the Emitters of the component (e.g. knowledge, information about the environment, behaviour, etc.), and propagates its adaptation policies and actions using eFfectors through the Controller ports of the managed component. The AM has the same structure of a SC, but it is conceived to be associated with a SC, and along with the SC, it composes one of the basic patterns. An AM cannot live alone because in its inside it explicates a feedback loop that needs an external component to manage (e.g. a SC or another AM). So in its internal the AM has the logic to monitor, analyse, plan and execute on the managed component. An example of an AM is an external component with a camera that is able to communicate with a system of robots and can manage the system giving information on the presence of obstacles in the environment, or on the presence of objects to carry. The AM is not able to directly act on the environment, but only on the robots (that are the one to act on the environment).

Figure 2 summarises the three basic types for individual components that are created starting from the basic components. Adding an external feedback loop to a component (either a SC-Reactive, or a SC-Proactive, or another AM), a new individual pattern can be created, as we can see in the taxonomy presented in Figure 3.

3.3 Mechanisms of Composition

In the previous subsection we considered the basic components that compose self-adaptation patterns at the *component* level, but of course self-adaptation patterns can be adopted also at *ensemble* level, by composing adaptive SCs. In order to rationally organize and describe all the self-adaptive patterns, we now introduce the mechanisms by which SCs can be composed into composite autonomic components or ensembles.

To this purpose, we can consider two primary dimensions for the composition (see Figure 3);

1. Composition at the *component* level, corresponding to viewing Figure 3 row by row, in particular on the first row from left to right. This is where an individual component (a SC-Reactive or a SC-Proactive) *is extended adding more and more autonomicity* with the aid of AMs.

2. Composition at the *ensemble* level, corresponding to viewing Figure 3 column by column, from up to down. This is where individual *components are brought together to form an ensemble* and in which different ways of organizing the interactions in the system correspond to different self-adaptation patterns[1].

In our taxonomy, we only focus on the interaction's mechanisms between components (SCs, AMs and also the environment) that aim at supporting a self-adaptive behaviour. We do not report here all the possible interactions between components (and also with the environment) if these interactions (e.g. negotiation, direct communication, etc.) are not directly related to enact adaptivity. For example, we know that all the components live in an environment, and usually communicate with/through it, but the environment is considered in our pattern only when it is a mean of adaptation (see the second row of the taxonomy table).

Adding autonomicity (with the aid of AMs) is one of the basic mechanisms of composing patterns – read the taxonomy table column by column –, along with the creation of systems by adding components – read the taxonomy table row by row. By applying these mechanisms we are able to obtain all useful self-adaptive patterns (represented with bold lines in Figure 3). Using these mechanisms more and more times (e.g. adding different levels of AMs), we can obtain more complex and elaborated self-adaptation patterns (e.g. the ones represented by light lines in Figure 3). We express these new patterns in terms of *composite* self-adaptation patterns, rather than as *basic* ones[2].

At the *component* level, starting from the basic component (SC-Reactive or SC-Proactive), we enable more autonomicity by adding external Autonomic Managers (AMs), which will explicit the feedback loop that expresses adaptivity. The AMs can be added in parallel (third cell of first row of Figure 3) or one on top of the other - hierarchy (fourth cell of first row of Figure 3). The SC-Reactive with an external AM and the SC-Proactive are very similar patterns. This is because in both cases the feedback loop is only one, but in the latter case it is implicit inside the component, as if the component has an internal AM; while in the former case it is explicitly using an external AM.

These patterns are both very useful because the pattern composed of SCs-Reactive and AMs permits a system to have simply components, and to add the autonomicity and

[1] As a general consideration, we emphasize that the number of components that take part of a specific self-adaptation pattern in an ensemble is not relevant. In fact, if the ensemble is dynamically determined by means of some "ensemble attributes", the arrival and dismissing of specific SCs, provided the overall structure of the self-adaptive pattern is preserved, is not relevant.

[2] The fact that these self-adaptation patterns represent the basic ones is not formally proved yet, although we are making some progress in this direction.

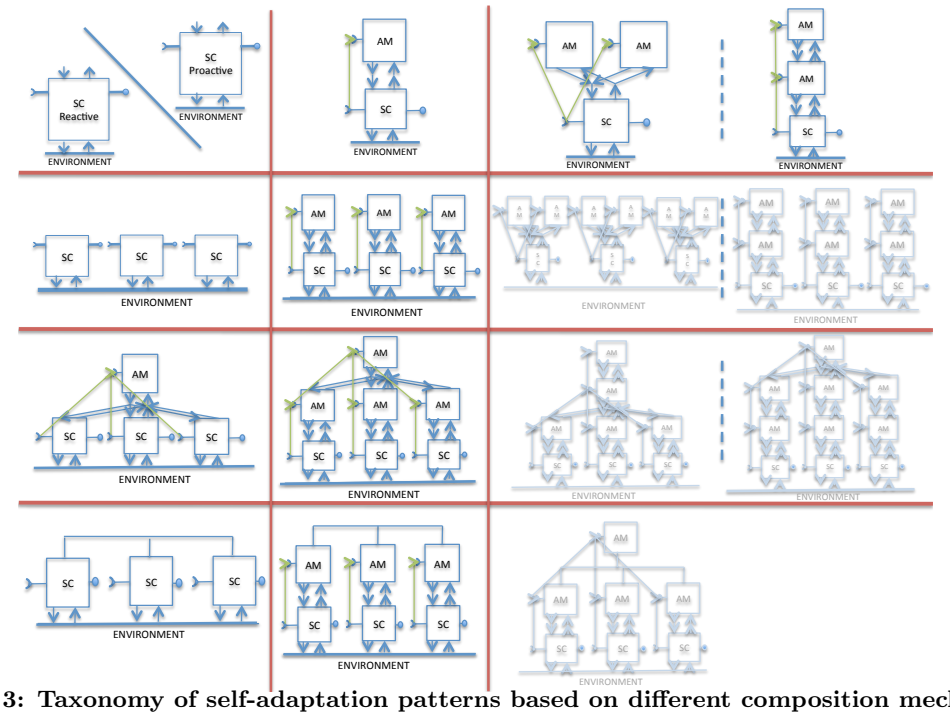

Figure 3: Taxonomy of self-adaptation patterns based on different composition mechanisms

the adaptation logic by the AMs. Instead the patterns of SCs-Proactive make it possible to use fewer components in an ensemble, but more complex ones that have their internal goals.

At the *ensemble* level we compose more basic components in order to create a system where each component has the same autonomic behaviour. Each SC behaves in the same way inside the ensemble. We describe the three main rows of the table:

1. The second raw considers ensembles of SCs in which the environment is considered as the means of adaptation and where the feedback loop that controls adaptation is implicit in the indirect (*stigmergic* [8]) interactions of the components via such shared environment. These self-adaptation patterns include all typical schemes of *swarm intelligence* [3]. The pattern can consider both SCs that are not autonomic (first cell of second row - Figure 3) as well as SCs that are made autonomic by means of AMs attached individually to each of them, but whose AMs do not coordinate with each other (second cell of second row - Figure 3);

2. The third row considers ensembles of SCs that are made autonomic by some organizations of AMs that explicitly coordinate the adaptation actions of the system. This can consider both a fully centralized coordination of adaptation (first cell of third row - Figure 3), in which a single AM controls all SCs, as well as a hierarchical organization in which individual AMs control individual SCs, and a central AM coordinates all AMs (second cell of third row - Figure 3);

3. The fourth row includes ensembles in which components directly coordinate and negotiate (in a fully distributed decision making approach) their adaptation actions with each others. Such direct negotiation of adaptation actions can take place between SCs, where it makes sense if they are SCs-Proactive (first cell of fourth row - Figure 3) or via direct communication acts of their AMs (second cell of fourth row - Figure 3).

3.4 The Taxonomy

To create a complete and rational taxonomy, we have defined the model of components that makes it possible to describe all the components used in the self-adaptation patterns. Then we have defined two mechanisms of composition that permit both to describe basic self-adaptation patterns, and to compose more complex patterns according to the developer needs. These mechanisms allow us to fill our taxonomy with the patterns that describe all the simplest adaptation mechanism, and then to extend the table with all the possible patterns for self-adaptive systems.

As we can see from Figure 3, the first row and the first column show how we can derive elementary patterns from the basic patterns (that are the first and second cells of first row). This can be done by adding a first level of autonomicicy (third and fourth cell of first row), or by adding components to create ensembles (second, third and fourth cells of first column).

Then, if we compose a pattern of the first row with the ones of the first cell of the next rows, we find out other elementary patterns (as shown in the taxonomy table). However we can see further on (section 4.4) that it is also possible to create a new pattern composing two or more cells of Figure 3. Therefore there is not the need to complete the taxonomy table and the catalogue of patterns (presented in the next section), because this would be a too huge effort and would be beyond our aim.

The presented composition mechanisms let the developer define a self-adaptive pattern, taking inspiration and expe-

riences from the basic ones and looking at how they perform adaptation in a system.

Another important aspect that emerges from the taxonomy table are common properties that derived from each row of the table. In the first row there are patterns that describe single components. But then, the patterns that are classified in the second row are used in systems that: (i) are composed of a large number of components (e.g. thousands of components); (ii) are composed of quite simple elements (elements that are not able to directly coordinate and adapt each others using direct communication mechanisms); (iii) are situated in environments that frequently change (the environment plays a key role in the system). While the patterns that are situated in the third row are used in systems that: (i) are composed of a limited range of components and where a central coordination is necessary. The central AM, due to the limited number of elements, it is not considered as a single point of failure; (ii) are composed of elements that are able to communicate to each other, but are not able to coordinate adaptation without a centralised AM that coordinates all the adaptation mechanisms; (iii) are situated in an environment that does not necessarily influence the system. The patterns that are situated in the fourth row are used in systems that: (i) are composed of a large number of components that need to directly communicate to enact adaptation but where a centralised AM is considered as a single point of failure; (ii) are composed of elements complex enough to manage adaptation on their own. The elements are able to communicate to each other the adaptation mechanism; (iii) are situated in environments where a direct communication between elements is preferable (e.g. a single change in the environment must condition all the elements in the same way).

4. EXEMPLARY PATTERNS

The presented taxonomy table is very useful to understand how self-adaptation can be performed into a system. Starting from this taxonomy, we have defined a complete description and characterisation of the single self-adaptation patterns by means of a *catalogue* of self-adaptation patterns. This catalogue aims at being a guideline for the correct choice of which the self-adaptation pattern to be applied in a system that must exhibit self-adaptive behaviours.

Moreover the common properties of each row of the taxonomy table aid to understand which category of patterns is the most appropriate regarding the system's requirements.

To help developers in their choice, every pattern needs to be described with useful and complete information, so we exploit a *pattern template*, as described in the next subsection.

4.1 Pattern template

We have defined a pattern template to describe the self-adaptation patterns of the taxonomy in Figure 3 so as to include all the information needed to support developers in understanding how and when to apply each self-adaptation pattern.

Tacking inspiration from consolidated literature, we have decided to adopt a template similar to that of the "basic design pattern" exploited in [9], but also absorbing self-adaptation-specific elements from [20]. Following Ramirez and Cheng we have dropped out the field "Known as" because it is not applicable for our scope, and the "Sample Code" field because it is too application specific. Instead of the latter we have added the field "Applications" that is useful to understand how the specific pattern can be applied. Moreover we also aim at using Unified Modelling Language (UML), or UML like diagrams to describe the pattern's interface, instead of the Object Modelling Technique (OMT) used by Gamma et al..

Clearly, to be correct and useful, the catalogue has to be based on experiences and/or on some solid formal ground, and on a solid organization. Therefore the catalogue we have defined reports a list of case studies that apply self-adaptation patterns in different real life scenarios.

In addition, a peculiar field that we have decided to add to our pattern template concerns the typical structure of the goals that a pattern can satisfy. In general terms, a catalogue of patterns is useful if, starting from the requirements of a system, the catalogue includes suggestions on which pattern fits which structure of requirements. To explain this, we should recall that, in autonomic systems, requirements are often expressed in goal-oriented terms [16]. Specifically, accordingly to the terminology adopted in the SOTA conceptual model [1], requirements are expressed in terms of *goals*, intended as desirable situations (or "state of the affairs") that the system (or its components) should *eventually achieve*, and *utilities*, intended as situations that the system should try to *continuously preserve*.

In goal-oriented analysis, the requirement analysis typically results in a list of *goals* and *utilities* associated to individual components, or to ensembles of components. There could be individual components (as well as ensembles) sharing some sub-portion of such goals and utilities, and there could be goals and utilities specifically associated to one component or one ensemble. A key question that arises is therefore: given the structure that the goals and utilities that my system-to-be will have to satisfy, which patterns better fit such structure? A catalogue of patterns should help finding an answer to the above question, which we try to accommodate in the specific "SOTA description" section.

All of this said, the structure of the pattern template that we have adopted is as follows:

- **Pattern Name**: Name that will identify the pattern.
- **Classification**: If the pattern is for SC or for Ensemble.
- **Intent**: A description of the problem the pattern addresses.
- **Context**: The conditions in which the pattern should be applied.
- **Structure**: A representation of the pattern interface in terms of UML/UML-like diagrams.
- **Behaviour**: A description/representation of how the pattern achieves its main objectives.
- **SOTA description**: A description of the structure of goals and utilities that fit this pattern. Specifically:
  - *Goals*: A formal description of the typical structures of the goals that the pattern can satisfy.
  - *Utilities*: A formal description of the typical structure of the utilities that the pattern can satisfy.

  - *Explanation*: Informal description and explanation of the rationale for the above structures.

- **Consequences**: A description of how objectives are supported by the given pattern and a list of the trade-offs and outcomes of applying the pattern.

- **Related Patterns**: Additional patterns that are commonly used in conjunction, and patterns derived from that, of from which this pattern derives.

- **Applications**: A list of use cases that apply this pattern.

In the next subsections we will show some patterns presented in the proposed taxonomy: the *Autonomic Service Component Pattern* (row 1 column 2), the *Reactive Stigmergy Service Components Ensemble Pattern* (row 2 column 1), the *P2P AMs Service Components Ensemble Pattern* (row 4 column 2), and some composed patterns. For the complete catalogue of self-adaptation patterns see [17].

Due to space limit, it is not possible to report in this paper examples of the selection and application of the right pattern in order to create an adaptive system. For further details see [18, 19].

4.2 Reactive Stigmergy Service Components Ensemble Pattern

- **Pattern Name**: Reactive Stigmergy Service Components Ensemble.

- **Classification**: Pattern for Service Components Ensemble.

- **Intent**: There are a large amount of components that are not able to directly interact one to each other. The components simply react to the environment and sense the environment changes.

- **Context**: This pattern has to be adopted when:

  - there are a large amount of components acting together;
  - the components need to be simple component, without having a lot of knowledge;
  - the environment is frequently changing;
  - the components are not able to directly communicate one with the other.

- **Structure**: See first cell of the second row at the taxonomy table in Figure 3.

- **Behaviour**: This pattern has not a direct feedback loop. Each single component acts like a bio-inspired component (e.g. an ant). To satisfy its simple goal, the SC acts in the environment that senses with its "sensors" and reacts to the changes in it with its "effectors". The different components are not able to communicate one with the other, but are able to propagate information (their actions) in the environment. Than they are able to sense the environment changes (other components reactions) and adapt their behaviour due to these changes.

- **SOTA description**:

  - *Goals*: $G_{SC_1}, G_{SC_2}, \ldots, G_{SC_n}$
  - *Utilities*: $U_{SC_1} = U_{SC_2} = \ldots = U_{SC_n}$
  - *Explanation*: In the pattern each SC has a separated goal, that is explicit only at the component level.
    Regarding the utilities of the ensemble, they are the same of each SCs that have to be shared by the components.

- **Consequences**: If the component is a proactive one, its behaviour is defined inside it with its internal goal. The behaviour of the whole system cannot be a priori defined. It emerges from the collective behaviour of the ensemble. The components do not require a large amount of knowledge. The reaction of each component is quick and does not need other managers because adaptation is propagated via environment. The interaction model is an entirely indirect one.

- **Related Patterns**: Proactive Service Component.

- **Applications**:

  - A case study that uses this pattern is the ant system for web search presented in [10]. This system is composed of a colony of cooperative distributed agents. The intelligent behaviour arises because of the agent's interaction with the environment, and indirectly with the other system's agents. Each agent corresponds to a virtual ant that has the chance to move itself from the hypertextual resource where it is currently located, to another URL. A sequence of links represents a possible agent's route, where the pheromone trail could be released on at the end of each exploration. The pheromone trails represent the means that allows the ants to take better local decisions with limited local knowledge both on environmental and group behaviour. The ants employ the pheromone to communicate the exploration result to another: the more interesting resource an ant was able to find out, the more pheromone trail it leaves on the followed path.
  - Another system that uses this pattern is described in [22]. Here Agents are situated in an environment, which they can perceive. Using this environment they can indirectly interact with one other. Agents are able to adapt their behaviour according to the changing circumstances in the environment. The overall application goals result from interaction among agents, rather than from capabilities of individual agents.

4.3 P2P AMs Service Components Ensemble Pattern

- **Pattern Name**: P2P AMs Service Components Ensemble.

- **Classification**: Pattern for Service Components Ensemble.

- **Intent**: This pattern is designed around an explicit autonomic feedback loop for each component. The

components are able to communicate and coordinate each other through their AMs. Each AM manages adaptation on a single SC.

- **Context**: This pattern has to be adopted when:
  - the components are simple and an external AM is necessary to manage adaptation at the component level;
  - the components need to directly communicate one with the other (through their AMs) to propagate adaptation.
- **Structure**: See second cell of the fourth row at the taxonomy table in Figure 3.
- **Behaviour**: Each component is managed by an AM and acts as an autonomic component. Than the AMs directly communicate one with the other with a P2P communication protocol. The communication made at the AM's level makes it easier to share not only knowledge about the components, but also the adaptation logic.
- **SOTA description**:
  - *Goals*: $(G_{SC_1}, G_{AM_1}) \bigcup (G_{SC_2}, G_{AM_2}) \bigcup \ldots \bigcup (G_{SC_n}, G_{AM_n})$
  - *Utilities*: $(U_{SC_1}, U_{AM_1}) \bigcup (U_{SC_2}, U_{AM_2}) \bigcup \ldots \bigcup (U_{SC_n}, U_{AM_n})$

  The goal of the ensemble is composed of the goals of every single component. Here a component is composed of a SC and an AM, so its goal is the goal of the SC (if it is a proactive component), along with the goal of the AM.
  At the same way the utilities of the ensemble are composed of the utilities of every single component. In this scenario, it is not necessary that all the components have the same utilities (same for goals), and also some components may have no utilities at all.
- **Consequences**: The use of AMs to communicate between components makes it more simple the adaptation management because the components remain simple and the knowledge necessary for adaptation is easily shared between the AMs.
- **Related Patterns**: Autonomic Service Component.
- **Applications**:
  - A lot of case studies about intelligent transportation systems use this pattern. For example a traffic jam monitoring system case study is presented in [2]. The intelligent transportation system consists of a set of intelligent cameras, which are distributed evenly along a highway. Each camera (SC) has a limited viewing range and cameras are placed to get an optimal coverage of the highway. Each camera has a communication unit to interact with other cameras. The goal of the cameras is to detect and monitor traffic jams on the highway in a decentralised way. The data observed by the multiple cameras have to be aggregated, so each camera has an agent that can play different roles in organisations. Agents exploit a distributed middleware, which provides support for dynamic organisations. This middleware acts as an AM; it encapsulates the management of dynamic evolution of organisations offering different roles to agents, based on the current context.
  - The same case study is presented in [12] that develops it with an Aspect-oriented architecture.

4.4 Example of composed pattern

After a description of the basic patterns, we present some case studies where we can clearly view the use of a composite pattern. These composite patterns are useful when the system's requirements present features that can be found in different patterns that, brought together better describe the system, than using only one of them.

The Autonomic Computing Reference Architecture [4] depicted in Figure 4, is composed of some *Centralised AM Service Component Ensemble Pattern* (see the red dots part), with another centralised AM. In the composition there is also the *Parallels AM Service Component Pattern* (see the yellow dashed line part). These starting patterns are taken by the patterns taxonomy: the basic pattern is the one of the second row, first column, where the SC can be replaced by the same pattern. The architecture is made by a hierarchical set of resource managers (that we identify as AMs), which manage a set of resources (SCs). Then an orchestrating manager (that we identify as a super AM) controls the management operations of the resource managers, and the resource managers provide the management support for a set of resources. In this architecture it is also possible that parallels AMs manage a resource. We remark that the SCs can be both Reactive and Proactive in the same system.

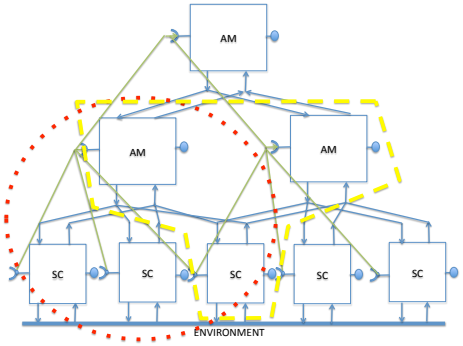

Figure 4: Composed pattern in the Autonomic computing reference architecture [4]

5. CONCLUSIONS

Adaptivity is a very important feature in today's systems, and there are several architectural choices to enact it in the developed systems, leading to different *self-adaptation patterns*. In this paper we have proposed a classification of such patterns by means of a taxonomy, which classifies them following two orthogonal composition mechanisms: at the *component* level and at the *ensemble* level. The proposed taxonomy outlines not only the features of the self-adaptation patterns, but also the connections among them.

Moreover, applying the proposed mechanisms allows us to have a taxonomy that is not fixed, but that can be expanded by means of composition, and new self-adaptation patterns can be considered as composition of existing ones.

This taxonomy has a twofold use. First, it can be exploited to classify existing adaptive systems, so to find out connections and common features with similar systems. Second, developers can exploit it to have an idea of the features of the existing self-adaptation patterns and to decide the most appropriate one to be chosen in the development of new systems.

With regard to future work, we are currently working in different directions. First, we aim at proving that the proposed taxonomy is complete, i.e. it covers *all* possible self-adaptation patterns. Second, we aim at including our taxonomy in a more general framework that describes the relationships among service components, autonomic managers and the environment. Third, we are exploring the chance of dynamically changing the adopted self-adaptation pattern in order to better face new runtime conditions; we call this property *self-expression*, because the systems are able to express themselves (i.e., to achieve their goal) independently of unexpected changes [25].

6. ACKNOWLEDGMENTS

Work supported by the ASCENS project EU FP7-FET, Contract No. 257414.

7. REFERENCES

[1] D. Abeywickrama, N. Bicocchi, and F. Zambonelli. Sota: Towards a general model for self-adaptive systems. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 48–53, Toulouse, France, 2012. IEEE.

[2] J. Andersson et al. Modeling dimensions of self-adaptive software systems. In B. Cheng et al., editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 27–47. Springer, 2009.

[3] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, USA, 1999.

[4] P. Brittenham et al. It service management architecture and autonomic computing. *IBM Systems Journal*, 46(3):565–581, 2007.

[5] Y. Brun et al. Engineering self-adaptive systems through feedback loops. In B. Cheng et al., editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 48–70. Springer, 2009.

[6] G. Cabri, M. Puviani, and F. Zambonelli. Towards a taxonomy of adaptive agent-based collaboration patterns for autonomic service ensembles. In *Conference on Collaborative Technologies and Systems*, pages 306–315, Philadelphia (USA), 2011. IEEE.

[7] B. Cheng et al. Software engineering for self-adaptive systems: A research roadmap. In B. Cheng et al., editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2009.

[8] M. Dorigo, E. Bonabeau, and G. Theraulaz. Ant algorithms and stigmergy. *Future Generation Computer Systems*, 16(8):851–871, 2000.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, 1995.

[10] F. Gasparetti and A. Micarelli. Swarm intelligence: Agents for adaptive web search. *ECAI*, pages 1019–1020, 2004.

[11] H. Gomaa and K. Hashimoto. Dynamic self-adaptation for distributed service-oriented transactions. In *International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 11–20, Zurich, Switzerland, 2012. IEEE.

[12] R. Haesevoets et al. Weaving the fabric of the control loop through aspects. In *Self-Organizing Architectures: First International Workshop*, volume 6090, pages 38–65. Springer-Verlang, 2010.

[13] S. Hariri, B. Khargharia, H. Chen, J. Yang, Y. Zhang, M. Parashar, and H. Liu. The autonomic computing paradigm. *Cluster Computing*, 9(1):5–17, 2006.

[14] IBM-Corporation. An architectural blueprint for autonomic computing. *Autonomic Computing White paper*, 36:34, 2006.

[15] W. M. et al. Ascens: Engineering autonomic service component ensembles. In *Formal Models for Components and Objects 2011, Post Proceedings*, volume 7542 of *LNCS*. Springer, 2012.

[16] M. Morandini et al. On the use of the goal-oriented paradigm for system design and law compliance reasoning. In *iStar 2010–4 th International i* Workshop*, page 71, Hammamet, Tunisia, 2010.

[17] M. Puviani. Catalogue of architectural adaptation patterns, 2012.

[18] M. Puviani. Self-expression in adaptive architectural patterns. *Awareness Magazine*, 2012.

[19] M. Puviani, G. Cabri, and L. Leonardi. Adaptive patterns for intelligent distributed systems: A swarm robotics case study. *Intelligent Distributed Computing VI*, pages 241–246, 2012.

[20] A. Ramirez and B. Cheng. Design patterns for developing dynamically adaptive systems. In *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 49–58, Cape Town, South Africa, May 2010. ACM.

[21] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2):14, 2009.

[22] D. Weyns, S. Malek, and J. Andersson. Forms: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems*, 7(1):8, 2012.

[23] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. Göschka. On patterns for decentralized control in self-adaptive systems. pages 76–107, 2012.

[24] M. Wooldridge. *An introduction to multiagent systems*. Wiley, 2002.

[25] F. Zambonelli et al. On self-adaptation, self-expression, and self-awareness, in autonomic service component ensembles. In *Self-Adaptive and Self-Organizing Systems Workshops*, pages 108–113, Ann Arbor, Michigan, USA, 2011. IEEE.