ascens

www.ascens-ist.eu

ASCENS

Autonomic Service-Component Ensembles

D4.1: First Report on WP4

Catalog of Patterns of Component- and Ensemble-Level Self-Adaptation and Self-Expression, and Requirements for Knowledge Modeling

Grant agreement number: **257414**
Funding Scheme: **FET Proactive**
Project Type: **Integrated Project**
Latest version of Annex I: **7.6.2010**

Lead contractor for deliverable: **UNIMORE**
Author(s): **Franco Zambonelli (UNIMORE), Dhaminda B. Abeywickrama (UNIMORE), Nicola Bicocchi (UNIMORE), Mariachiara Puviani (UNIMORE) Rosario Pugliese (UNIFI), Emil Vassev (UL)**

Due date of deliverable: **September 30, 2011**
Actual submission date: **November 15, 2011**
Revision: **Final**
Classification: **PU**

Project coordinator: **Martin Wirsing (LMU)**
Tel: **+49 89 2180 9154**
Fax: **+49 89 2180 9175**
E-mail: **wirsing@lmu.de**

Partners: **LMU, UNIPI, UDF, Fraunhofer, UJF-Verimag, UNIMORE, ULB, EPFL, VW, Zimory, UL, IMT, Mobsya, CUNI**

SEVENTH FRAMEWORK PROGRAMME

Executive Summary

In this report we summarize the work performed in WP4 during the first year of the ASCENS project, and the key results achieved. First, we frame the overall research approach that we have adopted. Then we introduce the SOTA ("State Of The Affairs") modeling approach, which we have defined as an innovative conceptual framework to model adaptation requirements, knowledge requirements, and to subsequently help designer in choosing the most appropriate adaption patterns. Following, we present our preliminary taxonomy of self-adaptation patterns and analyze the key concepts and mechanisms of dynamic self-expression. The ASCENS case studies have been extensively exploited to assess research activities, also via some early simulation experiences.

Contents

1 Introduction and Research Approach

One of the specific research objectives to be faced by ASCENS, and the specific focus of WP4, is to study, analyze, and experiment with the various models, schemes, and mechanisms via which autonomic self-adaptation can be expressed, at various levels, in service components (SCs) and in service component ensembles (SCEs). The ultimate goal is to provide a sound and uniform set of conceptual and practical guidelines and tools to guide developers of service component ensembles in the engineered exploitation of such mechanisms at the level of abstract system modeling, verification, and implementation.

The overall goal of WP4 is very ambitious. Indeed, the goal of studying adaptation in general foundational terms contrasts with the many application- and domain-specific adaptation needs that one can observe in different scenarios. The risk is that in missing the proper trade-off between generality and usability. On the one hand, one can approach the problem by trying to be extremely general, and thus missing in proposing conceptual and practical tools that can be effectively usable in specific application scenarios. On the other hand, one can approach the problem in a more practical way, e.g., by working on real-world examples, and thus being eventually unable to synthesize general-purpose foundational lessons. The research approach we have adopted has been shaped so to minimize such risks.

1.1 Research Approach

To tackle the above issue, the first year of activities in WP4 has adopted a very pragmatic approach, as illustrated in Figure 1. The idea has been to identify the many inter-related activities that are required to effectively guide towards the most appropriate identification of the adaptation requirements of a system and, thus, of the most appropriate architectural patterns by which such adaptation can be achieved. In particular, the adopted research approach (and the scientific/technical contributions that are currently being provided as a result of the activities) is as follows:

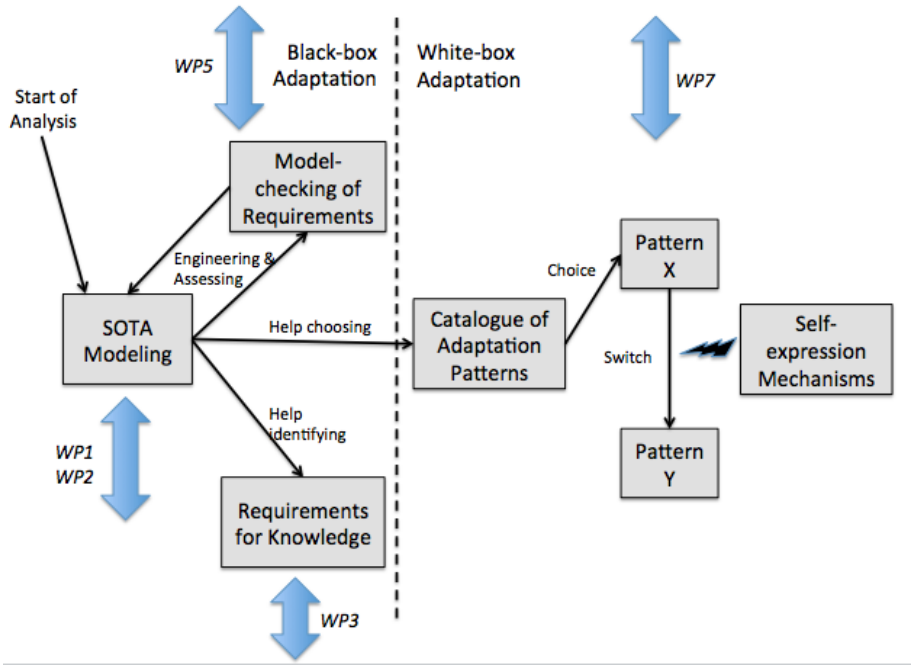

Figure 1: An overview of the research approach followed by WP4.

- We have tried to model in very general terms the key conceptual dimension of adaptation, expressed as an observable property of a software system. That is, we have adopted a sort of "black-box" approach to adaptation in which, abstracting from the actual mechanisms of adaptation and from domain-specific aspects of it, we questioned about "what adaptation is for" from the viewpoint of system requirements and observable behavior of a system. The result of this process is a robust conceptual and operational framework, SOTA ("State Of The Affairs"), that can be used to elicit and rationally represent adaptation requirements.

- SOTA can be used to early assess adaptation requirements via model-checking techniques, which is very important to verify such requirements well before any design choice is made on the systems. Also, SOTA can be used to help identifying the awareness requirements of a system, and thus as a tool by which to support the process of identifying which knowledge must be made available to the system and its components, which processing techniques should be applied to this knowledge, and which representation better suits such knowledge.

- SOTA enables expressing adaptation requirements without having to early commit to specific adaptation schemes (a key shortcomings of many approaches to the engineering of self-adaptive systems). However, and very important for our work in WP4, it represent a useful means to move towards a "white-box" approach to adaptation, in which choices should be made about the internal software structures and processes that one can adopt to achieve adaptation. That is, SOTA can be a means by which one can be guided on identifying the specific mechanisms and architectural patterns of self-adaptation, by focusing on what such mechanisms and patterns are for, other then simply on what they are.

- On these bases, we have started classifying and analyzing – at the level of both individual SCs and SCEs – the possible architectural schemes that can be adopted (and that have been often proposed in different areas) to express adaptive behaviors. The result of this activity, which will continue in the following years, will eventually be a rationally organized set of architectural *design patterns* [GHJV95], that can help designers in selecting the most appropriate adaptation solution for their problems.

- The self-adaptive patterns we analyze determine a sort of "weak", or first-order, adaptation [ST09]. To push adaptation capabilities farther, and make SCs and SCEs capable of adaptation beside the specific situations which their current pattern enable, there is need of a "strong", or second-order, adaptation: SCs/SCEs should be able to "dynamically self-express" on need the most suited architectural pattern of self-adaptation. Thus, we will also study the issue of dynamic self-expression and in particular the mechanisms that make it possible for SCs/SCEs to dynamically self-express a change in their structure. As of now, this study is can be considered at a preparatory stage.

1.2 Relations with other WPs

The adopted research approach clearly implies coordinating with other WPs, and helps positioning and relating with respect to them. In particular:

- The model checking of SOTA requirements clearly implies cooperating and harmonizing with the activities devoted to verification in WP5. Although a strong cooperation with WP5 was not originally planned, the possibility opened by the SOTA model as far as model checking is concerned, calls for a future tighter coupling of the WP4 and WP5 activities.

- The identification of knowledge requirements in SOTA will support the activities of WP3, where the issues of knowledge representation and modeling are dealt with [VHGN11]. With this regard, interaction and exchange of information have been already intense over the first year of the project, and has already led to a substantial agreement on model and engineering issues related to self-awareness and self-adaptation.

- The SOTA model has been defined and will be refined in strict cooperation with WP1 and WP2, where a language for SCs/SCEs and innovative formal models for adaptive systems [HW11], respectively, are being studied by taking into account the lessons of SOTA and, vice versa, by usefully feeding back our WP4 activities. In particular: two joint meetings with WP1 researchers have been already organized to verify that the WP1 language can support expressing the adaptation model of SOTA; frequent and intense discussions with WP2 researchers have taken place to ensure that the mechanisms and models there studied could have been effectively framed within SOTA and the self-adaptive patterns frameworks.

- All activities have been and will be performed by focusing on the practical application scenarios, as being studied in WP7. In particular, so far: the e-mobility case study have been adopted to perform modeling and experiments on model-checking requirements with SOTA; the robotics case study has been adopted for studying adaptation patterns.

1.3 Structure of the Document

The remainder of this document is organized as follows:

- Section 2 introduces the SOTA model and sketches its main characteristics and potentials;

- Section 3 sketches how the SOTA model can be made operational and can be used – as shown via examples applied to the e-mobility case study – to effectively model check the consistency of collected requirements;

- Section 4 shows how the SOTA model can be used to identify requirements for knowledge modeling and representation;

- Section 5 introduces our early taxonomy of adaptation patterns, details a few patterns we have identified, and discusses their language-related issues;

- Section 6 analyses the key mechanisms we have identified so far to enable self-expression;

- Section 7 sketches the early simulation experiments we have performed on the robotics case study.

Eventually, Section 8 summarizes and details the future plan of activities.

2 Black-box Adaptation: What Adaptation is For?

The key motivations for studying adaptation are that: *(i)* we need to build a system to achieve some functionalities; but *(ii)* there are contingent situations that may undermine the capability of achieving such functionalities, because the system is immersed in an open-ended and dynamic environment; thus, *(iii)* means must be devised for the system to be able to achieve its functionalities despite contingencies.

Accordingly, the starting point for understanding and framing adaptation concept is necessarily in the modeling of the requirements of the system (i.e., analyzing adaptation from a "black-box" viewpoint abstracting the actual mechanism via which adaptation can be achieved), in order to understand both what the system should achieve, what are the contingencies that may prevent it to achieve, and what are the mechanisms to achieve despite contingencies.

To this end, we have defined SOTA as a conceptual model to support the analysis of requirements for complex self-adaptive, and the later architectural design. The SOTA conceptual model will be naturally complemented by more elaborated formal models for adaptive ensembles, as being developed developed in WP2 coherently with SOTA, and early described in [HW11].

2.1 The SOTA Goal-oriented Modeling Approach

The SOTA conceptual model builds on most assessed approaches to goal-oriented requirements engineering [TPYZ09, MSS+10], and integrates and extends them with recent approaches on multidimensional context modeling [RLS+11]. Such generalization and extension try to account for the general needs of dynamic self-adaptive systems and components.

In general terms, a goal is a specific "state of the affairs" (from which the SOTA acronym derives) that has to be reached by an entity. The capability of pursuing a goal naturally matches goal-oriented and intentional entities (e.g., humans, organizations, and multiagent systems), and consequently autonomic and self-adaptive systems.

Interestingly, goal-oriented modeling of self-adaptive systems enables a uniform and comprehensive way of modeling functional requirements and non-functional ones, the former representing the eventual state of affairs that the system has to achieve, and the latter representing the current state of the affairs that the system has to maintain while achieving. For systems that are not goal-oriented in nature (e.g., systems that simply encode some functionality or service) functional requirements can be expressed as a NULL goal, while the need for adaptivity reduces to expressing those goals that represent non-functional requirement. In the following we will talk of "goals" when talking about functional requirements, and will talk about "utilities" to refer to non-functional requirements (although other may suggest adopting the term "constraints").

In SOTA, the "state of the affairs" represents the state of everything in the world in which the system lives and executes that may affect its behavior and that is relevant w.r.t. its capabilities of achieving. We could also say that such state of affairs is the "context" of the systems. In particular, the current "state of the affairs" $S(t)$ at time t, of a specific entity e (let it be an SC or an SCE) can be described as an n-tuple of s_i values, each representing a specific aspect of the current situation:

$$S(t) =< s_1, s_2, \ldots, s_n >$$

As an example, an e-mobility scenario includes a number of entities such as users, vehicles and parking lot operators, all of which immersed in a multidimensional SOTA space that include dimensions such as: parking space availability, charging lot availability, climate comfort level, other than dimensions related to the street and traffic situation.

As an entity e executes, its position S changes either due to the specific actions of e or because of the dynamics of e's environment. Thus, we can generally see this evolution of S as a movement in a virtual n-dimensional space $\mathbf{S}$ (see Figure 2):

$$\mathbf{S} =< \mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_n >$$

Or, accordingly to the standard terminology of dynamical systems modeling, we could consider $\mathbf{S}$ as the phase space of e and its evolution (whether caused by internal actions or by external contingencies) as a movement in such phase space.

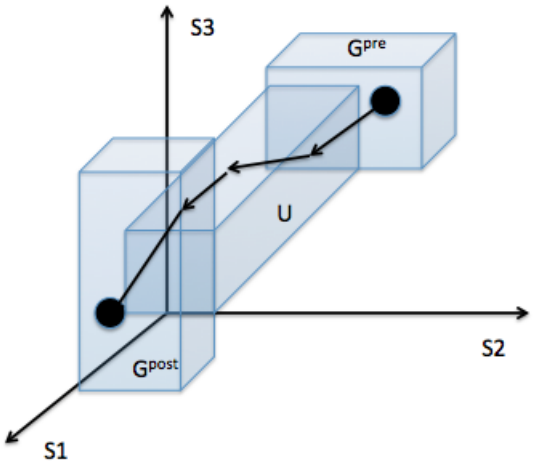

Figure 2: The trajectory of an entity in the SOTA space, starting from a goal pre-condition and trying to reach the post-condition while moving in the area specified by the utility.

To model such evolution of the system in terms of "transitions", $\theta(t, t+1)$ expresses a movement of e in the $\mathbf{S}$, i.e.,

$$\theta(t, t+1) =< \delta s_1, \delta s_2, \ldots, \delta s_n >, \delta s_i = (s_i(t+1) - s_i(t))$$

A transition can be endogenous, i.e., induced by actions within the system itself, or exogeneos, i.e., induced by external sources. The existence of exogeneos transitions is particularly important to account for, in that the identification of such source of transitions (i.e., the identification of which dimensions of the SOTA space can induce such transition) enables identifying what can be the external, non controllable, factors requiring adaptation.

A **goal** by definition is the eventual achievement of a given state of the affairs. Therefore, in very general terms, the specific i^{th} goal G_i for the entity e can be represented as a specific point in such space or, more in general, as a sub-space of $\mathbf{S}$, i.e.:

$$G_i =< A_1, A_2, \ldots, A_n >, A_k \subseteq \mathbf{S}_k$$

Getting more specific, a goal G_i got an entity e may not necessarily be always active. Rather, it can be the case that an entity has a goal activated only when specific conditions occur. In these cases, it is useful to characterize a goal in terms of a "pre condition" and of a "post-condition", to express when the goal has to activate and what the achievement of the goal implies, i.e., $G_i = \{G_i^{pre}, G_i^{post}\}$ where both G_i^{pre} and G_i^{post} represent two areas (or points) in the space $\mathbf{S}$. In simple terms: when an entity e finds itself in G_i^{pre} the goal gets activated and the entity should try to move in $\mathbf{S}$ so as to reach G_i^{post}, where the goal is to be considered achieved (see Figure 2). Clearly, a goal with no pre-condition is like a goal whose pre-condition coincide with the whole space, and it is intended as a goal that is always active.

Goals represent, to most extents, functional requirements. However, in many cases, a system should try to reach its goals by adhering to specific constraints on how such goal can be reached. By referring again to the geometric interpretation of the execution of an entity as movements in the space $\mathbf{S}$, one can say that sometime an entity should try (or be constrained) to reach a goal by having its trajectory confined within a specific area (see Figure 2). We call these sorts of constraints on the

execution path that a system/entity should try to respect " utilities", to reflect a nature that is similar to that of non-functional requirements.

A utility U_i can be typically expressed as a subspace in $\mathbf{S}^e$:

$$U_i^e =< A_1, A_2, \ldots, A_n >, A_k \subseteq \mathbf{S}_k$$

and can be either a general one for a system/entity (the system/entity must always respect the utility during its execution) or one associated to a specific goal G_i (the system/entity should respect the utility while trying to achieve the goal). For this latter case, the complete definition of a goal is thus:

$$G_i = \{G_i^{pre}, G_i^{post}, U_i\}$$

In some cases, it may also be helpful to express utilities as relations over the derivative of a dimension, to express not the area the trajectory should stay in but rather the "direction" to follow in the trajectory (e.g., try to minimize execution time, where execution time is one of the relevant dimension of the state of affairs). It is also worth mentioning that utilities can derive from specific system requirements (e.g., for an e-vehicle, trying to reach a destination while minimizing fuel consumption) or can derive from externally imposed constraints (e.g., trying to reach a destination in respect of existing speed limitations).

A complete definition of the requirements of a system-to-be thus implies identifying the dimensions of the SOTA space, defining the set of goals (with pre- and post-condition, and possibly associated goal-specific utilities) and the global utilities for such systems, that is, the sets:

$$\mathbf{S} =< \mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_n >$$
$$\mathbf{G} = \{G_1, G_2, \ldots, G_n\}$$
$$\mathbf{U} = \{U_1, U_2, \ldots, U_n^e\}$$

In the e-mobility scenario, all the entities such as users, vehicles and parking lot operators, can be modeled in terms of entities having goals (e.g., a vehicle would like to reach a destination and park close to it in short time). Also, utilities can be identified related to how such goals can be achieved. Some of these utilities could be at the level of individual component goals (e.g., reaching a destination with appropriate climate comfort level in the vehicle itself) or at the global system level (e.g. minimization of overall pollution or traffic jams).

2.2 Using SOTA

Let us now shortly outline why SOTA is useful, and its definition has been indeed a key preparatory activity for WP4.

- A sound requirements engineering activity is necessary to pave the way for the effective development of a system. SOTA, by gathering from the lessons of goal-oriented requirements engineering and from modeling approach to context-aware systems, appears very suitable with this regard. In particular, SOTA supports a simple operational model that makes it possible to adapt and apply existing model-checking techniques to goals and utilities, and thus assess and improve requirements identification. This issue is described in Section 3.
- The identification of the SOTA space facilitates identifying the awareness requirements of the system-to-be. In fact, for a system to be autonomic, i.e., adapt and self-manage its execution autonomously, it must know whether it is acting in respect of goals and utilities. Thus, it must

have knowledge of its current and predicted position in the SOTA space for those dimensions of the SOTA space that are of relevance for goals and utilities. As a consequence, SOTA modeling can be useful to derive the knowledge and awareness requirements of the system. This issue is discussed in section 4.

- SOTA is a very practical conceptual framework to help understanding how, i.e., according to which scheme, a system should be architected to as to facilitate it in adaptively achieving its goals. That is, depending on the goals and utilities expressed during the requirements engineering phase, and depending on whether these are global or to be locally associated to components, it is possible to be guided in the choice of the most suitable architectural patterns, both at the level of components and ensembles. This specific issues is analyzed in Section 5, along with the presentation of our early taxonomy of adaptation patterns.

3 Model Checking SOTA Goal-Oriented Requirements

The previous section presented the conceptual model for SOTA. In this section, by turning the conceptual model into an operational one, we show how SOTA can be an effective tool to perform an early, goal-level, model checking analysis for adaptive systems.

In our work, operational software requirements are derived systematically from the underlying goals and utilities. They contain tasks for goals and utilities modeling (e.g. refinement and decomposition into requirements) and their operationalization (e.g. identifying the events sequence, pre-conditions and post-conditions). Our approach essentially integrates and leverages the benefits of goal-oriented requirements elaboration with formal analysis techniques on event-based systems, two techniques that can be effectively integrated in a single approach [LKMU08]. In particular, our model checking approach is based on an operational, event-based, untimed and asynchronous model of labeled transition systems.

The advantages of the proposed approach are related to exploiting the goal-oriented modeling of SOTA. It provides a systematic method for modeling real-world goals of a system, such as a refinement hierarchy, conflicts and exceptions handling, thus gradually deriving specifications that satisfy the goals. Last but not least, the approach can exploit event-based systems for automating formal analysis of software architecture specifications, and for supporting software architecture design and program verification and testing. More specifically:

- Model checking can be used to verify whether a set of required pre-conditions and post-conditions forms a complete operationalization of a goal or utility.
- Model checking can be used to check the satisfaction of higher-level goals. This allows the requirements engineer not to confine verification to a single goal or utility but a portion of the goal graph (i.e. multiple goal or utility operationalization).
- Any inconsistencies and implicit requirements can be detected as deadlocks.
- Animation of the goals-oriented models can be performed using the standard animation and simulation features of the LTSA.

The current model checking approach is based on formal verification, validation and simulation techniques provided by the model checker Labeled Transition System Analyzer (LTSA) [MK06] and its process calculus Finite State Processes (FSP) [MK06]. The formalism that we use to model goals and utilities is the Fluent Linear Temporal Logic (FLTL) of the used model checker. Fluents have

been used to provide a uniform framework for specifying properties that combine event and state-based predicates, and to automatically verify their satisfaction by an event-based model. Our operational model was motivated by the formal Tropos language [FMPT01] and the KAOS methodology [vLDDD91].

The approach is explored and validated using the e-mobility case study (individual planning vehicles scenario [HWB+11]) of the ASCENS project, adapted to the goal-oriented requirements modeling domain. An extended technical report [AZ11] is available that contains full details of the approach and an extensive elaboration of the e-mobility case study. Full details on the e-mobility case study can be found in deliverable D7.1.

3.1 Approach Overview

Our overall model checking process (Figure 3) is divided into four main stages.

First, early requirements are described using the well-assessed i* framework's [MSS+10]. The i* framework is centered on three concept types: actors, intentional elements and intentional links. Actors are active entities of the organizational context who perform actions to achieve their goals, for example vehicle drivers and parking lot operators in e-mobility. We use a strategic dependency model to describe the dependency relationships among various actors. In the i* framework, there are several types of intentional elements that can act as dependencies: *goals*, *soft goals*, *tasks* and *resources*. We further extend this to include *utilities*. These represent the intentional elements of our i* model. In our operational model, a goal is a functional requirement of the state of the affairs an actor eventually aims to *achieve*, for example assigning a car park space. On the other hand, a utility represents a non-functional requirement of the state of the affairs, which is required to be *maintained* or *avoided* by an actor while achieving a goal. Examples of utilities are avoiding low battery levels or maintaining climate comfort while reaching the destination. Exemplary goals and utilities for the e-mobility case study are illustrated in Figure 4.

Second, and since the i* model only addressed static aspects, goals and utilities are represented into an operational SOTA language, to describe the dynamic aspects of dependencies among SCs and SCEs. The syntax of such SOTA language adopts a context-free grammar, which consists of a number of productions. To represent the dynamic characteristics, actors, entities and dependencies are represented as classes in the SOTA language. On the one hand, the grammar provides information on the class description defining the structure of the instances with their attributes. Attributes (constants or variables) of a SOTA class are used to represent relationships between objects. The satisfaction of goals and utilities can be expressed using three goal patterns: *achieve*, *maintain* and *avoid*. In general, a goal will have the *achieve* pattern while a utility will have the *maintain* or *avoid* pattern. On the other hand, the grammar defines properties expressed in typed first-order linear-time temporal logic formula. Properties, which can be pre-conditions and post-conditions, are used to describe the dynamic aspects of actors, entities and dependencies. Domain properties provide descriptive statements about the environment, which can be physical laws or organizational policies. We provide case study examples of the grammar, instantiating it to classes.

Third, a methodology to map or translate SOTA to event-based, asynchronous FLTL of the LTSA for formal model checking purposes is provided. To this end, each actor or entity, which can be an SC/SCE from the object model, is modeled as a finite state process in FSP. Then the bounding of the SOTA goal-oriented model is performed and fluents are identified. This involves transforming a SOTA goal-oriented model that has an infinite state space into a finite state fluents-based SOTA model. On this base, initiating and terminating events for each fluent are defined, and the pre-conditions and post-conditions for goals and utilities are modeled as asynchronous FLTL assertions. In general, goals are modeled as liveness properties while utilities are modeled as safety properties in FLTL. Finally,

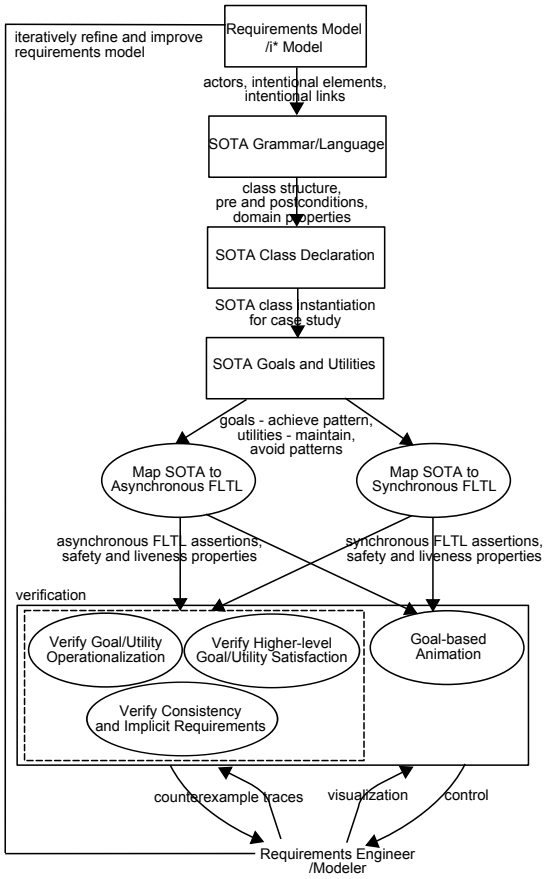

Figure 3: Model checking SOTA goal-oriented requirements: the overall approach.

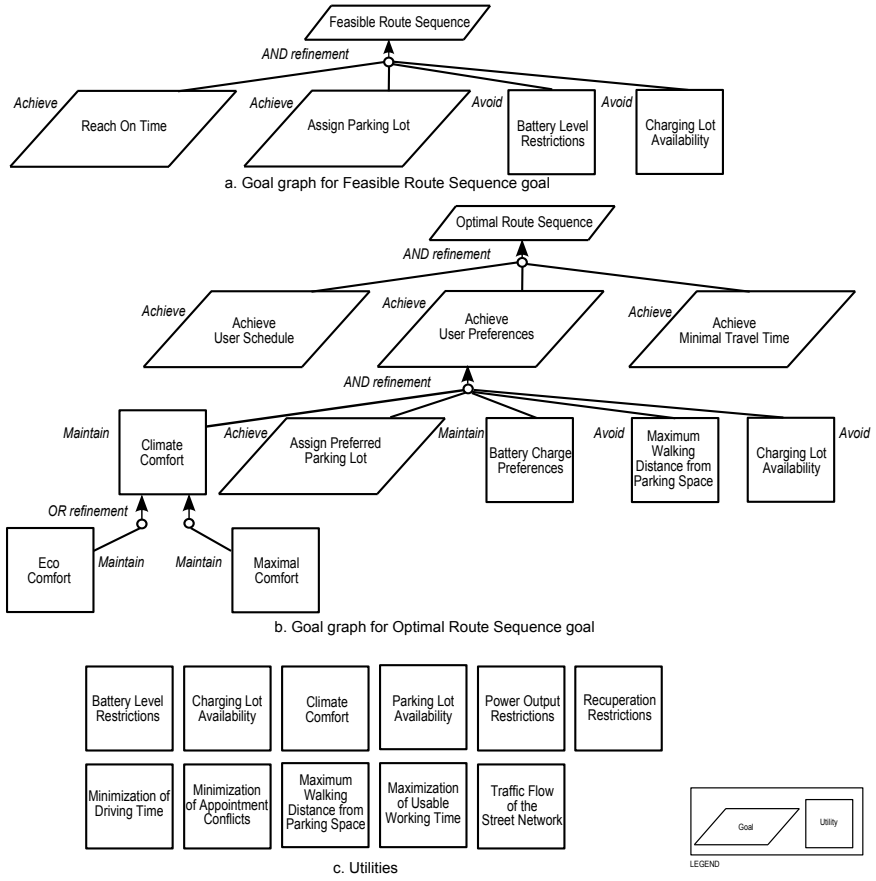

Figure 4: Exemplary goal graphs and utilities for the e-mobility case study modeled according to the i* framework.

```
1 const N=4 // number of car spaces in the car park
2 PARKINGSPACECONTROL(N=4) = SPACES[N],
3 SPACES[i:0..N]           = (when(i>0) arrive->availableParkingLots[i-1]->SPACES[i-1]
4                             |when(i<N) depart->availableParkingLots[i+1]->SPACES[i+1]).
5 VEHICLEARRIVALS   = (arrive->VEHICLEARRIVALS).
6 VEHICLEDEPARTURES = (depart->VEHICLEDEPARTURES).
7 ||CARPARK = (VEHICLEARRIVALS||PARKINGSPACECONTROL(4)||CHARGINGLOTSCONTROL(2)||VEHICLEDEPARTURES).
8 assert G_POST_ASSIGNPARKINGSPACE = []<>availableParkingLots[i:1..N]
9 ...
10 // The pre-condition (U_CHARGINGLOTSAVAILABLE) checks whether there is a charging lot
11 // available in the car park before considering availability of parking space.
12 assert G_ASSIGNPARKINGSPACE_CHARGINGLOTSAVAILABLE = (U_CHARGINGLOTSAVAILABLE && G_POST_ASSIGNPARKINGSPACE)
13 ...
14 // A higher-level goal with the use of the && operator
15 assert G_ASSIGNPARKINGSPACE_CLIMATECOMFORT = (U_CLIMATECOMFORT && G_POST_ASSIGNPARKINGSPACE)
```

Figure 5: FSP code for the ASSIGNPARKINGSPACE goal.

the LTS model of the software-to-be is obtained by composing the component behavior models with the FLTL assertions defined for the goals and utilities.

In the fourth stage of our model checking process, we verify the asynchronous FLTL assertions derived for the goals and utilities. Model checking is performed: to identify any *incompleteness* of the SOTA goal-oriented requirements model; to identify *inconsistencies* and *implicit requirements* that can be detected as deadlocks; and to *animate* the goals-oriented models using the standard animation and simulation features of the LTSA. The counterexample traces generated in the model checking process can iteratively refine and improve the SOTA requirements model, thus deriving a reliable and robust requirements specification. This formal analysis is also important for the white box framework and for the architectural patterns as it can help identify which patterns to adopt.

3.2 Verification: an Example

In this subsection we examine the actual verification with a case study example from e-mobility. The key goal of the example reported is to exemplify some key advantages of the approach. A more complete analysis of the case study can be found in [AZ11].

To illustrate goal or utility operationalization and higher-level goal satisfaction, an example of a goal modeled in the e-mobility case study is provided next. In the description, first, the name of the goal or utility is provided with a keyword–*Achieve*, *Maintain* or *Avoid*–which specifies the temporal pattern of the goal or utility. In general, the *Achieve* pattern is associated with a goal while the *Maintain* and *Avoid* patterns are identified with a utility. Next the goal or utility is described using natural language to facilitate communication with any stakeholder. This is followed by a formal definition of the goal or utility using first-order linear-time temporal logic notation. An LTSA model checker-based FLTL code is provided next to define the goal or utility for verification purposes. Finally, the results of model checking are discussed with any counterexamples generated.

Goal Achieve[ASSIGNPARKINGSPACE]
Def. Assigns a parking space to an e-vehicle. At least one charging lot needs to be available at the car park before checking whether a parking space will eventually be available.
FormalDef. $[]\neg(\text{carpark.availableChargingLots} = 0) \wedge \Diamond(\text{carpark.spaces} \geqslant 0)$

The VEHICLEARRIVALS and VEHICLEDEPARTURES processes model the arrival and departure of vehicles to the car park (ln 5–6, Figure 5). The PARKINGSPACECONTROL process models a car park with four parking spaces (ln 2–4). It allows cars to enter the park when there is at least one

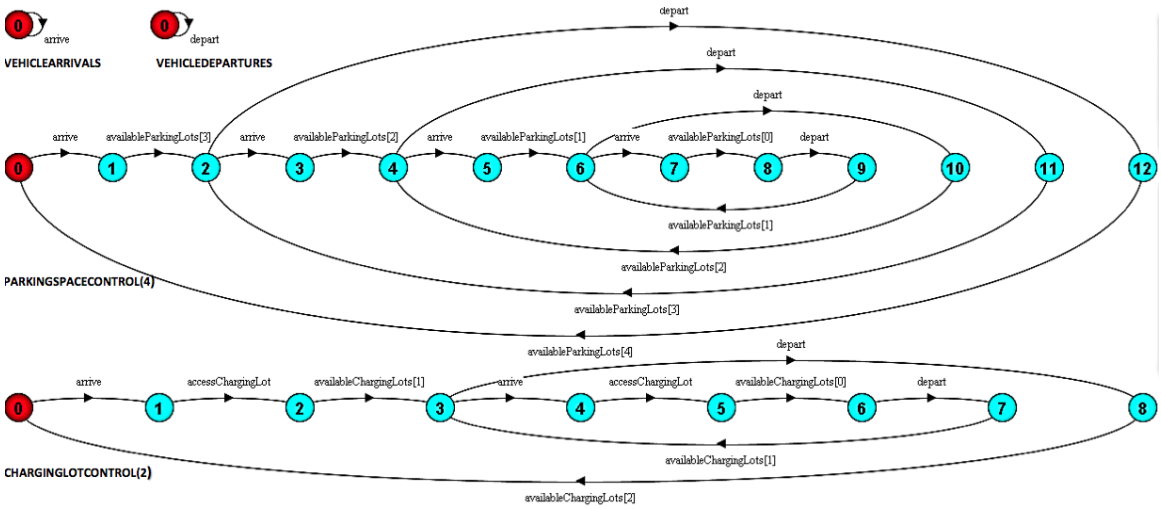

Figure 6: LTS model for the ASSIGNPARKINGSPACE goal.

space available.

The CARPARK is the composed process of vehicle arrivals, departures, parking space control and charging lots control. The post-condition for this goal has been defined as a simple liveness property, which asserts that it is always the case that a parking space will eventually be available (G_POST_ASSIGNPARKINGSPACE, ln 8). The pre-condition for this goal is the utility U_CHARGING-LOTSAVAILABLE, which specifies that at least one charging lot needs to be available at the car park. The automatons generated from this property are provided in Figure 6.

The automaton for G_ASSIGNPARKINGSPACE_CHARGINGLOTSAVAILABLE corresponds to the negation of the property as required by model checking. It illustrates that * actions are required in state 0. These actions allow the automaton to non-deterministically move to the acceptance state. As there is parking space available there is no counterexample trace generated for the post-condition of the goal. Ln 14–15 provide an example of a multiple goal operationalization with the use of the && operator. G_ASSIGNPARKINGSPACE_CLIMATECOMFORT is a higher-level goal concerned with achieving user preferences (Figure 4). Goal decomposition is performed by refining it through the conjunction of U_CLIMATECOMFORT and G_POST_ASSIGNPARKINGSPACE.

A typical problem that can occur in goal-oriented modeling is that an *inconsistency* or an *implicit pre-condition* can result in a deadlock in the specification. An inconsistency in the specification can occur for several reasons. For example, if the post-condition of a goal does not imply its pre-condition then the system might be in a state where the post-condition is true but the pre-condition is not. So the goal needs to be satisfied but it is not, thus leading to an inconsistency.

To illustrate this, let us consider the assign parking space goal discussed previously. A pre-condition was defined for this goal, which is the utility U_CHARGINGLOTSAVAILABLE (ln 10–12, Figure 5). This utility specifies that at least one charging lot needs to be available at the car park. The pre-condition checks whether there is a charging lot available in the car park before considering availability of parking space. There can be a situation where there is a parking space available (post-condition satisfied) but no charging station (pre-condition not satisfied). As a consequence, the goal needs to be satisfied but it is not, thus leading to an inconsistency.

On the other hand, implicit pre-conditions occur due to interactions between requirements on different goals. An implicit pre-condition occurs in the following scenario. A post-condition of a goal may implicitly prevent another goal being applied even if all the pre-conditions for the second goal are true. Such a situation can cause a deadlock in the specification if we do not model additional

constraints or properties to avoid it. Nevertheless, there is a benefit associated with implicit preconditions as it allows requirements engineers to derive a more reliable specification.

The goal-oriented requirements models of SOTA can be animated using the standard animation and simulation features of the LTSA tool (for example, see Figure 6). This can be used to explore model behaviors interactively with the users and other stakeholders. Counterexample traces generated can be used to iteratively refine and improve the SOTA requirements model, thus deriving a reliable and robust requirements specification.

4 Modeling of the SOTA Space and Knowledge Requirements

The "state of the affairs" is a very general concept, and its dimensions include anything relevant to keep a system up and running. Specifically, these can include hardware characteristics (e.g., available memory, load of CPU, network bandwidth, available sensors and devices), software characteristics (e.g., value of internal parameters, state and availability of related software components, current state of inter-components interactions, access to shared resources), as well as environmental characteristics (temperature, whether conditions, location and speed, state and availability of specific physical resources and objects, etc.).

Accordingly, prior to analyzing goals and requirements, a key issue in SOTA concerns acquiring a proper understanding of the domain-specific state of the affairs. This includes the phases of **Identification**, **Virtualization**, and **Metrification**.

4.1 Identification

First, it is necessary to understand which characteristics are relevant, considering both goals and utilities of the observed system. For both of them, areas $A_i \subseteq \mathbf{S}_i$ can be expressed in terms of conditional expressions over the values in $\mathbf{S}_i$. In the case, one of the dimensions S_i are not relevant towards the achievement of goals or towards the utilities, then for such goals and utilities $A_i \equiv \mathbf{S}_i$.

Indeed, in many practical cases, goals and utilities involves only a limited fraction of the n dimensions of the overall state of affairs space. That is, G_i is expressed as a set of points or areas in an m dimensional space (with $m < n$), projection of the original space, disregarding what happens in some of the $\mathbf{S}$ dimensions. If we consider a base vector:

$$B =< b_1, b_2, \ldots b_n >, b_i \in \{0, 1\}$$

such that

$$b_i = 0 \Leftrightarrow \forall G_i \in \mathbf{G} \wedge \forall U_i \in \mathbf{U} \longrightarrow A_i \equiv \mathbf{S}_i$$

then goals and utilities can be expressed in the sub-dimensional space:

$$\mathbf{SS} = B \times \mathbf{S}$$

The sub-dimensional space SS is important because it defines what information is relevant for the system, and which ones can be disregarded. That is, it drives the requirements for which knowledge has to be acquired, modeled, and made available to SCs and SCEs to enable adaptivity.

In addition, one should also account for specific contingent situations of the SOTA that may affect the capability of a system of achieving its goal in respect of the utilities, and that are not explicit in either $\mathbf{G}$ or $\mathbf{U}$. It may be necessary to identify these contingencies, identify when and how they could affect the capability of the system, and turn these explicitly either as utility functions (if we want to

find ways for the system to handle such situations in any case) or as "impossibility areas", i.e., areas of the SOTA space in which the system, however self-adaptive, will no longer be able to achieve.

As an example, consider a robot whose goal is to explore and map an environment with (as utility) the maximum possible speed will not have "temperature" as a natural dimension of the SOTA. However, if we imagine the robot finding itself exploring a building in fire, then we can chose: to continue ignoring this contingent state of the affairs and let the robot get burned, or to integrate temperature as an additional dimension in the SOTA space and define a utility function associated with the exploration goal that makes the robot stay away from high temperature areas. To most extent, if we assimilate the identification of goals and utilities to the "use cases" of traditional software engineering, the identification of such additional situations can be thought as the identification of the "alternative" situations in use cases.

So far, we assume that all dimensions in $\mathbf{S}$ are independent from each other; that is, a movement in $\mathbf{S}_i$ does not affect the positions in the other dimensions $\mathbf{S}_j$. Unfortunately, this is not always the case: the very characteristics of the domain can induce domain-specific constraints on how the movements can occur in such space. For instance, speed and residual battery of a modern electric vehicle are clearly related. Changing speed implies a change in battery's duration. Therefore, along with the identification of the goals and utility sets $\mathbf{G}$ and $\mathbf{U}$, it can be useful to identify constraints on the SOTA dimensions and on the "trajectories" that a system can follow on them.

4.2 Virtualization

Each of the identified relevant dimensions of the SOTA space implies the need for the SC/SCE to acquire information about its current positioning within the space. That is, there must exist some actual sensors to acquire the corresponding information. However, this is not a challenging issue *per se*: a number of different sensors are available to measure the most diverse features: from hardware-related to software-related ones, from those related to events of the physical world, to those related to the social world.

The key issue, instead, is that of providing SCs and SCEs with an appropriate view of what's happening, i.e., leveraging the typical low-level and punctual perspective of the actual sensors into that of a "virtual sensor", capable of providing an appropriate view representation of the values in that dimension, specifically tuned for the needs of the SCs/SCEs under analysis.

As an example, with reference to the e-mobility scenario, consider a system that is devoted to assist drivers as they drive. One of the dimensions of the SOTA space for such system can be the current attention level of the person. As of today, even the simplest smart phone embeds accelerometers to monitor the current patterns of movement of a user. However, actual accelerometers report movements in terms of raw numeric time series, whereas for the activity dimension of the SOTA space to be usable it would better consider activities as represented in terms of values such as "looking ahead", "sleeping", "looking rear". That is, as if a virtual sensor existed capable of reporting activity data in such terms.

Another important aspect of the virtualization process is that it detaches the provisioning of the virtual information from that of the actual sensors. For instance, it becomes irrelevant for an SC or SCE to know which actual sensor feeds virtual sensors. Consider the case of the human activity dimension. While the most obvious choice for producing such virtual information are smart phone accelerometers, one could also adopt also image processing techniques to provide such information, or even couple image processing with accelerometers to fuse the contributions of both into a more reliable composed virtual sensor.

In general, virtual sensors are useful for: *(a)* grouping a number of physical sensors for the sake of fault tolerance; *(b)* transparently converting low level sensor readings into semantically relevant in-

formation; and *(c)* grouping different physical sensors allowing multi-modal recognition capabilities.

The issue of identifying, during the modeling of a system, which kinds of virtual sensors are required to enable and facilitate adaptation, is thus necessary to properly drive activities related to knowledge modeling and processing, the latter required to turn physical sensors into virtual ones.

4.3 Metrification

The above process of virtualization implicitly carries the definition of the granularity of transition. That is, a transition for a component e, as previously defined:

$$\theta(t, t+1)_e =< \delta s_1, \delta s_2, \ldots, \delta s_n >, \delta s_i = (s_i(t+1) - s_i(t))$$

takes actually place and is perceivable by the component e itself only if some of the δs_i are different from 0 according to the virtual sensors values.

Hence, once the virtual sensor dimensions of the SOTA space are identified (and thus the granularity of transition), it may be necessary to associate a "metric" to the values in that space, to enable SC and SCE assessing whether they are properly aiming towards goal and towards the respect of the utilities.

As an example, a simple dimension such as temperature, may require to be mapped on different scales depending on the application scenario. A robot, to avoid get burned, may properly use a virtual temperature sensor with a scale of 10C. An driver-assistance system, on the other hand, may require a virtual temperature sensor with a scale of 0,1C.

In many cases (e.g., for the temperature, whatever the virtual scale) dimensions can be expressed as metric spaces of numerical values. In case of nominal values, instead, the issue of defining a proper metric (i.e., which concept is close to which other, and how a transition between concept can take place) arise. For instance, it is not straightforward to define a metric among high level concepts such as user activities. In some cases, it is possible to convert them into numeric values (e.g. $standingstill = 0$, $walking = 1$, $running = 2$, $biking = 3$, $driving = 4$) while in other cases the dimension has to be considered discrete.

4.4 From WP4 to WP3: Knowledge Requirements

All the above consideration about the modeling of the SOTA space eventually ends up in identifying in a quite accurate way the needs for the knowledge that a system has to achieve and how to properly adapt. In general terms, the modeling of the SOTA space, helps clearly separating the phases related to the modeling (and later implementation) of the system component to the one related to the modeling (and later implementation of associated processing and representation techniques) of the knowledge that they have to manage.

From the viewpoint of the ASCENS project, the modeling of the SOTA space is the natural intersection between the activities of WP3 and WP4: the identification phase is naturally in charge of WP4, the metrification is naturally in charge of WP3, while the issue of virtualization has to be necessarily addressed as a joint WP3/WP4 activity.

Further to this, the following of our analysis, and in particular the "white box" one, could help identifying further requirements for the knowledge dimension, in particular as far as the issue of whether knowledge should be locally to some SCs or globally to some SCEs. For example, an initial conclusion is that a SC may have three levels of knowledge: *(i)* a complete self-knowledge that encompasses functionalities, internal structure, abilities, goals, policies, local services, execution history, current situation, activities, intentions, etc.; *(ii)* incomplete but sufficient knowledge of its ensemble in terms of common goals, ensemble states, communication mechanisms and interfaces, neighboring

SCs, groups, etc.; and *(iii)* a rather general understanding of its operational environment in terms of concepts, objects, events and situations. Further detailed requirements for knowledge representation and reasoning have been elaborated in [VHGN11], and include:

- knowledge models: knowledge should be structured in four knowledge models: SC, SCE, environment and situations;
- knowledge specification and storing: knowledge is a formal specification of knowledge data stored by SCs and retrieved by inferential engines;
- ontology support: the ontology approach should be used to structure knowledge;
- general and factual knowledge: facts, rules and constraints should be used to allow for general and factual knowledge about predicates, names, connectives, quantifiers and identity;
- distributed reasoning: knowledge might be shared for reasoning involving multiple SCs;
- situations: situations should be elaborated as a knowledge entity;
- policies: behavior policies might be used to drive the SCs in situations;
- experience: SCs experience need to be automatically stored and retrieved for reasoning purposes.

5 White-box Adaptation: Adaptation Patterns

The SOTA modeling approach is very useful to understand and model the functional and adaptation needs of a system, as well as to check the correctness of specifications.

However, when switching from the "black-box" requirements analysis to the actual "white box" design of a system, the issue arise of identifying which architectural schemes should be chosen for individual SCs as well as for SCEs. With adaptivity in mind, the goal of such choice is to make sure that the adopted architectural scheme can support the capability of the system of self-adapt its behavior without undermining its functional behaviors.

5.1 Rationale for the Analysis of Adaptation Patterns

The way we have initially undertaken the study of self-adaptation for SCs and SCEs have been by identifying the key architectural patterns that could be adopted to enforce self-adaptation (other than self-expression) at the level of individual components and of ensembles. The result of this early analysis is extensively described in [CPZ11]. The rational underlying it can be summarized as follows:

- The capabilities of self-adaptation in a system necessarily requires the existence of *control loops*, as it is widely recognized in the area of self-adaptive systems [BDMSG$^+$09, CdLG$^+$09, VWMA11]. Control loops implies that, somehow, there exist means to inspect and analyze what is happening in the system (at the level of SCs, or SCEs, or at the level of the environment in which they situates) and have components of the systems react accordingly.
- Therefore, framing the possible architectural means by which such feedback control loops are integrated in the system (whether integrated explicitly by design or emerging implicitly from interactions) can be a suitable approach to identify self-adaptation patterns and reason on them.

5.2 Taxonomy

The taxonomy of patterns that we have originally identified is schematically represented in Figure 7. Without entering here into details (for which we forward the reader to [CPZ11], we can see that:

- At the level of individual components, and beside non-adaptive primitive service components (i.e., simply capable of providing functions) the three main self-adaptive patterns categories are: *(i)* reactive service components that are not coupled with an explicit control loop, but such control loops exists only implicitly in their interactions with the environment (as in reactive agent and component systems [BBD$^+$06]); *(ii)* components that have an internal control loop to direct their goal/utility-oriented behavior (as in intelligent and goal-oriented agents [BBD$^+$06]); *(iiI)* autonomic components explicitly coupled with an external control loop that monitor and direct their behavior (as in most of autonomic computing architectures [KC03, HKC$^+$06]).

- At the level of ensembles, the three key pattern categories are: *(i)* ensembles for which the overall adaptive activities are not explicitly engineered by design, but for which adaptiveness (and control loops) emerge from the interactions of the components with a shared environment (as in phermone-based [BCD$^+$06, KT11] and field-based approaches [MZ09]); *(ii)* ensembles in which the overall adaptive behavior is explicitly designed by means of specifically conceived interaction patterns between components (e.g., choreographies or negotiations [BS97, JFL$^+$01]), and in which mutual interactions implies the existence of control loops; *(iii)* ensembles in which there exists a set of components or "coded behaviors" that have the explicit goals of enforcing a global control loop over the ensembles, i.e., of controlling and directing their overall behavior (as in coordinated systems and electronic institutions [ERAS$^+$01]).

In addition, as from Figure 7, patterns at the component level are not orthogonal to the ones at the ensemble level, e.g., it is not possible to adopt a negotiation patterns in a ensemble of reactive components. This consideration is very important in that it can be useful to help designers in comprehensively attacking the choice of the most suitable adaptation patterns at both the component and ensemble level.

5.3 Patterns and SOTA

From what already said, a catalog of patterns makes sense if it can guide designers in making choices. However, after the preliminary analysis we have reported about (and which we consider very useful to frame some key ideas) we found it difficult to proceed with the activities to produce a complete catalog of self-adaptive patterns. In fact, for the catalog to be complete and capable of providing guidelines, it had to be based on experiences and/or on some solid formal ground, neither of which we had at the beginning of the project.

To this end, beside experiences on the case studies that will be extensively performed during the second year of the project (and that have already started [Puv11]), we have studied how SOTA modeling could provide advantages in the identification and modeling of patterns.

To understand what it could means to model patterns in terms of SOTA consider these two minimal examples of (incomplete) patterns description.

Reactive Service Component

- *Characteristics*: Reactive Service Components are characterized by the capability of perceiving (a portion of) the SOTA space and of applying actions to modify its position in it in the medium term (that is they have utilities). However, they do not have means to aim in the long term, that is, they do not have goals.

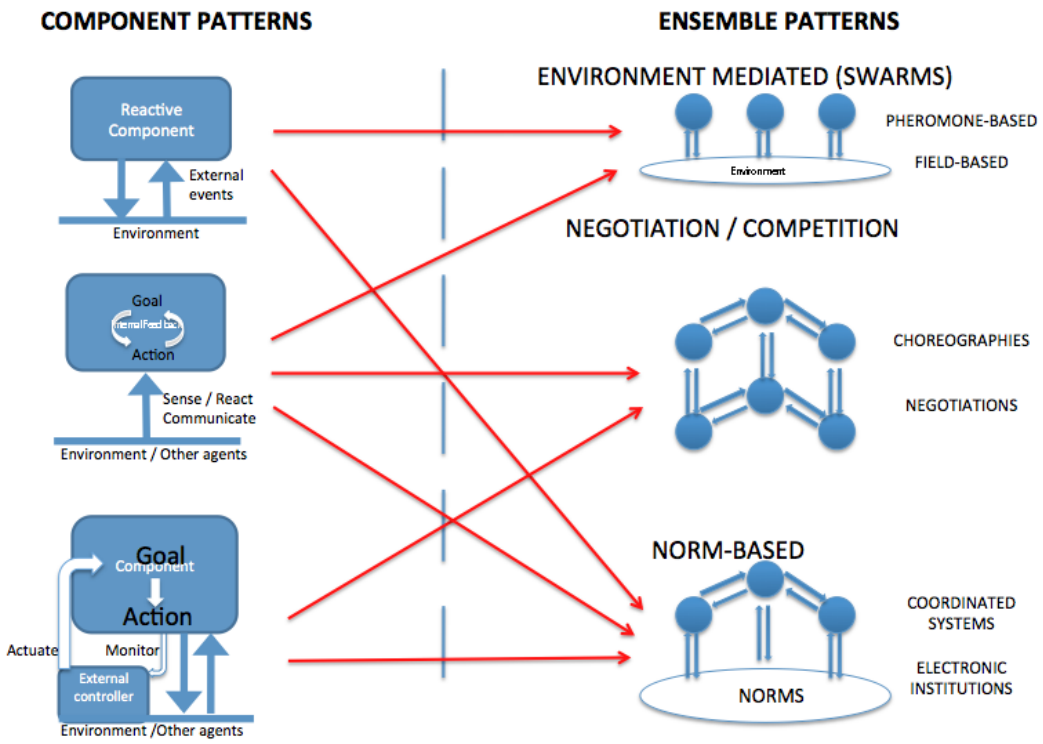

Figure 7: An overview of self-adaptation patterns.

- *SOTA Model*: $\mathbf{G} = \{\emptyset\}, \mathbf{U} = \{U_1, U_2, \ldots, U_n\}$
- *When to Adopt*: As a general rule, this pattern has to be adopted for a component when:
  - There is need for the component to ensure specific non-functional requirements in the provisioning of its services but the goals are expressed at a higher-level of the overall system, and cannot be mapped into individual goals
  - External events are frequent and need a rapid adaptation.
- *Examples*: Examples of components that use this pattern can be found in ant-based foraging systems [BDT99] and motion coordination [MMTZ06].

Goal-oriented Service Component

- *Characteristics*: Goal-oriented Service Components are characterized by the capability of perceiving (a portion of) the SOTA space and of applying actions to modify its position in it in the medium term (that is, they can have utilities) as well as in the long terms (that is, they have goals).
- *SOTA Model*: $\mathbf{G} = \{G_1, G_2, \ldots, G_m\}, \mathbf{U} = \{U_1, U_2, \ldots, U_n\}$
- *When to Adopt*: As a general rule, such pattern has to be adopted when:
  - There are components that are goal-oriented in nature (e.g., robots or navigator-enabled cars);
  - There are components that, despite being simply service-oriented in nature, they actively try to adapt their behavior, even without waiting for external call (or stimuli).

- *Examples*: An example of the use of this pattern is Jadex [BPL05], that supports cognitive agents by exploiting the BDI model. Other examples are goal-oriented systems, like robots [KBM98]. Another example of architecture that describe the component using this kind of patter can be the Rainbow architecture [CPGS09].

In both this examples, as preliminary as they can be, one can see that reasoning in terms of SOTA can facilitate both the description of the patterns and the understanding of when to adopt them.

In addition, from the analysis and model checking of the overall set of SOTA goals and utilities for a system to be, further hints can be derived for how (i.e., via which patterns) to architect the overall system. Although this study is only at the beginning, we can give a feeling of how this could happen.

Consider that, after the SOTA analysis and its model checking, one find that there exists two goals G_i and G_j that are "conflicting", i.e.,

$$G_i^{pre} \cap G_j^{pre} \neq \emptyset \wedge G_i \cap G_j = \emptyset$$

Then it is clear that if both these goals have to be pursued at the same time by the system, the designer is necessarily forced to decompose the system into two SCs (or SCEs), each of which devoted to independently try to reach the goal.

Consider instead that the analyst identifies the existence of two goals G_i and G_j that are "compatible" with each other, i.e.,

$$G_i^{pre} \cap G_j^{pre} = \emptyset$$

Then in this case the designer can evaluate to collapse the achievement of these two goals into a single one, to be assigned to a single goal-oriented SC (or SCE). In alternative, if the analysis shows that these two goals assume in any case the existence of two different SCs, the design can evaluate the possibility of realizing these two SCs with a simple reactive patterns and collapse the two goals into a single one, whose adaptive achievement is left in charge of a single external controller for the two SCs.

As preliminary and simple as these examples can be, they show the potentials of SOTA modeling in design, other than in analysis.

5.4 Adaptation Mechanisms and the ASCENS Language

For all the above adaptation patterns, the existence of a control loop implies the possibility for SCs or SCEs system to "understand" when and how to adapt. That is, to understand in which condition adaptation should take place and to put the appropriate adaptation actions at work, However, when it comes to implementation, adaptation mechanism must be put in place, and there must be means for designers and programmers to specify the actual adaptation mechanisms in simple and expressive terms.

In recent years, Aspect-Oriented Programming and Context-Oriented Programming have emerged as very flexible and promising approaches for programming self-adaptive software systems [GPS10, GPS11]. The idea is to make available language constructs and mechanisms by which to specify – within software components – which "situations" require adaptation and to dynamically activate the most appropriate behaviors (i.e., methods or algorithms) in response to them.

Accordingly, a necessary activity in the study of self-adaptive patterns, described in detail in [GLPT12], has been to:

- Understand how self-adaptive patterns could be effectively modeled within a tuple-based coordination language, to assess the coherence of the identified patterns and of the ASCENS language being defined in WP1;

- Understand how the adaptation mechanisms typical of AOP and COP can be effectively rendered within the ASCENS language under definition.

Such study, other than helping to better understand the relations between adaptation and programming languages, forms a necessary ground on which to rely to improve and refine our pattern catalog, i.e., to include examples of realization of each patterns in the ASCENS language.

6 White-box Adaptation: Towards Self-expression

As from previous discussion, a self-adaptation pattern specifies a specific architectural design for SCs or SCEs, with a specific accent on how the control loop enabling adaptation is integrated in them. A specific choice for the engineering of such control loops can make a system capable of adapting to a hopefully large, yet necessarily limited, range of contingencies.

Accordingly, a stronger, sort of meta-level, form of structural adaptation may be required, which is what we define "self-expression". Whenever the patterns adopted in a system (for some of its SCs or SCEs) appears do not longer adequate to properly and/or effectively support adaptation, a re-engineering of the structure of the system may be required, to re-shape the control loops that ensure adaptation.

6.1 Examples of Dynamic Self-expression

We have analyzed with more details the issue of self-expression in [ZBC+11]. Here, to clarify, we shortly summarize a there reported example of self-expression applied to the robotics case study.

Let us assume to have a large group of robots in charge of collectively explore and map an unknown environment. In this case, and given the availability of many robots, it is possible to think at organizing the group as a "swarm", and exploit collective intelligence to reach the goal. That is:

- Have each robot be programmed as a simple reactive entity (at least for its tasks related to exploring and mapping), i.e., without associating to each of them a control loop;
- Have the coordination among robots be based on repulsive pheromone trails (as in ant foraging [BDT99]), i.e., have an implicit set of control loops embedded in the mediated pheromone-based interactions that ensure adaptivity [BCD+06].

However, let us now assume that, for some unforeseen reasons, many of the robots dies (e.g., ran out of batteries earlier than expected due to high humidity rates). In this case, the swarm pattern (which requires a large number of robots to be effective) can soon turns to be ineffective, and a pattern based on direct goal-oriented negotiation between robots becomes be more suitable. This requires the group of robots to change their pattern of interactions (to switch from a swarm-based pattern to a negotiation-based one) and the robots themselves to switch from acting as simple reactive entities to acting as rational goal-oriented entities.

In a sentence: both the robots and the ensemble of robots have to self-express different patterns to re-engineer the structure of the control loops, in order to preserve effective adaptive capabilities.

6.2 Basic Mechanisms for Dynamic Self-expression

The study of self-expression in the first year involved, beside clarifying the issue by means of examples, identifying the basic mechanisms by which such self-expression can be actuated, either at the

level of SCs or of SCEs. Such basic mechanisms then form the basis from cataloging the patterns that, applying such mechanisms, enables an SC or an SCE to switch from one adaptation pattern to another.

Concerning the basic mechanisms, these have to be mechanisms that enables a dynamic change in the architecture of a component or ensemble. In the area of self-adaptive software [ST09], the issue of identifying such mechanisms have been extensively studies [ADLMW09, RC10], and there is a general agreement that the basic mechanisms include:

- Internalization. An individual software component (or a software system) internalizes some new component or behavior to change its basic internal architecture;
- Dismission. Vice versa, a component (or a software system) dismisses some of its components or behavior.

When contextualizing this basic mechanisms to the specifics of adaptive SCs and SCEs, and with a specific focus on the architecture of control loops, we obtain (for the level of individual SCs) the following mechanisms:

- Attachment of control loop, e.g., a reactive service component becomes a component explicitly coupled with an external control loop.
- Internalization of control loop, e.g., an autonomic service component becomes a goal-oriented component with an internal control loop.
- Externalization of control loop, e.g., a goal-oriented service component with internal control loop externalize its control loop to become an autonomic component;
- Dismission of control loop, e.g., a goal-oriented service component dismisses its control loop to become a reactive service component.

Basically, from the viewpoint of SOTA modeling, this mechanisms imply dismissing, internalizing, or externalizing, the issue of controlling the achievements of specific goals and utilities (see Figure 8).

Switching from the mechanism viewpoint to the pattern one, a pattern of dynamic self-expression expresses a movement from one adaptive pattern to another one by applying the necessary mechanisms. With reference to Figure 8, a possible pattern is the one in which a goal-oriented service component becomes a reactive service component by applying a mechanism of dismission of its internal control loop, and thus dismissing its responsibility over the taking care of goals and utilities.

At the level of SCEs, similar considerations could apply. However, it is clear that the application of self-expression mechanisms at the level of SCEs cannot abstract from applying (a set of) mechanisms at the level of the individual SCs. Also, we emphasize that dynamic changes in the structure of the control loops devoted to control the achievement of goals and utilities necessarily affect the kind of knowledge that components have to manage. That is, externalizing or internalizing control loops necessarily implies externalizing or internalizing some knowledge.

Over the second year, the comprehension and formalization of self-expression patterns will proceed in parallel with the study and experimentation of the self-adaptive patterns to which they apply, and of the implications of this on knowledge management.

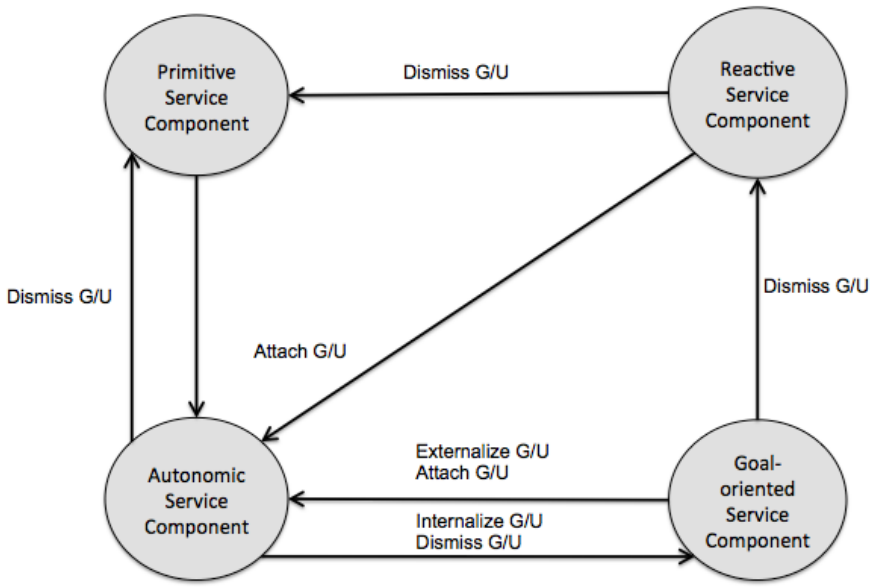

Figure 8: Patterns of individual dynamic self-expression.

7 Robotic Simulations

To better understand what adaptation means in practical terms, how adaptation can be performed both at level of SCs and at level of SCEs, as well as to enrich our pattern catalog with specific guidelines for their adoption, it is important to ground on practical experiences.

To this end, we have started experiencing with the robotics case study, in particular with the support of the ARGoS open source robot simulator. The key goals of our first experiences have been to unfold the idea of black-box adaptation and to pave the way for a better understanding of patterns.

The scenario we choose to kick off experiences [KB00] is aimed at understanding how each robot can contribute to system-level adaptation, and consider a group of robots in an environment. Each robot has the individual goal of finding some objects (sorts of virtual "food" items) in the environment and, carrying one at a time, move them back to a specific location of the environment. At the level of robot ensembles, though, a global goal (better, in SOTA terms, a global "utility") is that of preserving the overall level of batteries in the group of robot so as to ensure that the goal of finding and moving all objects can be achieved.

Starting from this scenario we already performed a set of different simulations, whose detailed description can be found in [Puv11], of which here we sketch only some key points.

In Figure 7 it is represented the area in which robots (situated in the lower part of the area, and acting as a virtual "nest" for the robot) normally reside, and where objects are (black dots) randomly distributed in it. The goal for robots is to periodically go out of the nest to find object to get back to it.

To shortly discuss a simple experiment, we have experienced a strategy in which, the robots of the group, after having spent some time out of the nest for finding food, rest in the nest for a time proportional to the time they have wondered around before finding food. The idea is: the more difficult is to find objects, the more is better to rest, not to have the whole group exhaust its batteries. As it can be seen in Figure 10 (where the X axis represents the simulation time and the Y axis represents the number of robots currently searching for food), such a simple strategy make the average number of active robots in the group adaptively change depending on the amount of existing food items (FI).

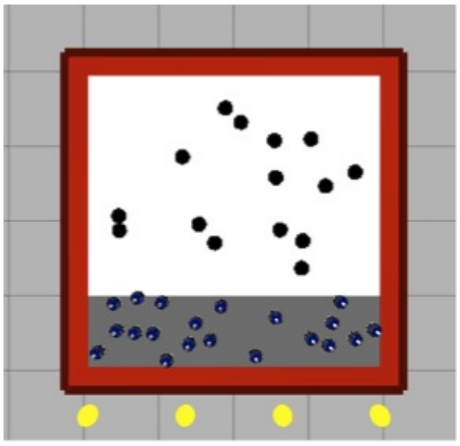

Figure 9: The arena of the robotic simulation.

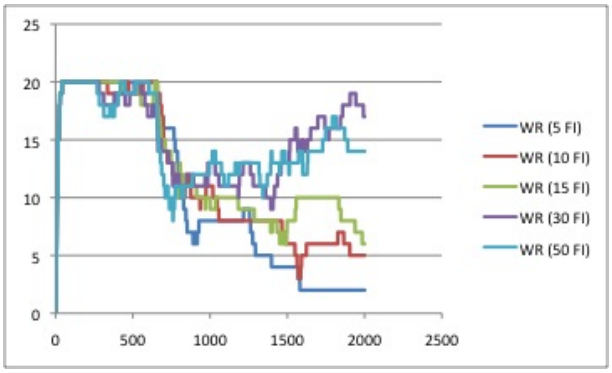

Figure 10: Results from a simple simulation experience.

8 Summary and Next Steps

Let us now summarize the key results achieved, and sketch the plans for the next year activities, also outlining how they relate with the other WPs of the project.

8.1 Summary

Overall, the first year of the activities within WP4 has brought a substantial amount of interesting scientific results. Specifically:

- We have defined the SOTA model, which can act as a sound comprehensive methodological foundation for studying self-adaptation and self-expression.
- We have shown how the SOTA model can be an effective tool for enabling early checking of requirements in complex autonomic software systems, and can interestingly more strongly relate the activities of WP4 with those of WP5.
- Again based on the SOTA model, we have defined guidelines for helping in identifying the requirements for knowledge modeling and identification. This activity has been performed in strict relation with WP3.

- We have defined a frame for classifying and identifying the key patterns of self-adaptation and self-expression in autonomic service component ensembles, and modeled a set of key relevant patterns. In addition, in cooperation with WP1, we have analyzed the issue of expressing such patterns via a suitable programming language for adaptive systems.

- We have started some initial experiences on the ASCENS case studies.

8.2 Plans for Next Year Activities

The activity of the first year has solidly paved the way for a smooth continuation of the activities in the next year, as well as for a tighter integration of the activities with the other WPs. In particular:

- Perform extensive simulations, and some real-world experiments, on the ASCENS case studies, to gather feedbacks and knowledge related to the behavior of the identified self-adaptive patterns, and eventually complete the pattern catalog. This work is expected to be carried on in strict cooperation with WP1, as far as the rendering of these patterns in the ASCENS languages is concerned, and with WP2, to start experiencing also with patterns based on advanced (e.g., game theoretic) mechanisms.

- Directly related, analyze how and in which conditions self-expression patterns can be effectively applied. This will require continuing the analysis of the WP7 case studies and of use cases within.

- Analyze how and to which extent the SOTA model, and in particular a requirements analysis based on SOTA, can facilitate designers in choosing and engineering the most suitable self-adaptive patterns for a system to be. This activity will be performed in strict cooperation with WP3, as far as the modeling of the associated knowledge is concerned, and with WP5, to be possibly supported by model checking techniques.

References

[Abe08] D. Abeywickrama. Pervasive services engineering for soas. In *ICSOC, International Conference on Service Oriented Computing, PhD Symposium*, page 29, Paphos, Cyprus, December 2008.

[ADLMW09] J. Andersson, R. De Lemos, S. Malek, and D. Weyns. Modeling dimensions of self-adaptive software systems. In B.H.C. Cheng, R. de Lemos, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 27–47. Springer, 2009.

[ADM05] D. Ancona, D. Demergasso, and V. Mascardi. A survey on languages for programming bdi-style agents, 2005.

[AHDJ01] P. Anthony, W. Hall, V.D. Dang, and N. Jennings. Autonomous agents for participating in multiple online auctions. In *Proc. of the IJCAI Workshop on EBusiness and the Intelligent Web*, pages 54–64, Seattle WA, USA, August 2001.

[AZ11] D. B. Abeywickrama and F. Zambonelli. Model checking SOTA goal-oriented requirements. ASCENS Project Technical Report No. 01, October 2011.

[BB05] A. Barros and E. Börger. A compositional framework for service interaction patterns and interaction flows. In *Proceedings of the Seventh International Conference on Formal Engineering Methods (ICFEM'2005)*, pages 5–35, Durham, UK, October 2005. Springer Verlang.

[BBD+06] R.H. Bordini, L. Braubach, M. Dastani, A.E.F. Seghrouchni, J.J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci. A survey of programming languages and platforms for multi-agent systems. *Special Issue: Hot Topics in European Agent Research II Guest Editors: Andrea Omicini*, 30:33–44, 2006.

[BCD+06] O. Babaoglu, G. Canright, A. Deutsch, G.A.D. Caro, F. Ducatelle, L.M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, et al. Design patterns from biology for distributed computing. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 1(1):26–66, 2006.

[BCD+07] P. Brittenham, R.R. Cutlip, C. Draper, B.A. Miller, S. Choudhary, and M. Perazolo. It service management architecture and autonomic computing. *IBM Systems Journal*, 46(3):565–581, 2007.

[BDMSG+09] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Engineering self-adaptive systems through feedback loops. In B.H.C. Cheng, R. de Lemos, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 48–70. Springer, 2009.

[BDT99] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, USA, 1999.

[Bil04] E.A. Billard. Patterns of agent interaction scenarios as use case maps. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(4):1933–1939, 2004.

[BP10] L. Baresi and L. Pasquale. Live goals for adaptive service compositions. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 114–123, Cape Town, South Africa, May 2010. ACM.

[BPBK04] P. Bresciani, L. Penserini, P. Busetta, and T. Kuflik. Agent patterns for ambient intelligence. In P. Atzeni, W. Chu, H. Lu, S. Zhou, and T.-W. Ling, editors, *Conceptual Modeling–ER 2004*, volume 3288 of *Lecture Notes in Computer Science*, pages 682–695. Springer, 2004.

[BPL05] L. Braubach, A. Pokahr, and W. Lamersdorf. *Jadex: A BDI-agent system combining middleware and reasoning*, pages 143–168. Springer, 2005.

[BS97] C. Beam and A. Segev. Automated negotiations: A survey of the state of the art. *Wirtschaftsinformatik*, 39(3):263–268, 1997.

[CBC09] R. Charrier, C. Bourjot, and F. Charpillet. Study of Self-adaptation Mechanisms in a Swarm of Logistic Agents. In *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 82–91, San Francisco, California, USA, September 2009. IEEE Computer Society Press.

[CdLG+09] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, et al. Software engineering for self-adaptive systems: A research roadmap. In B.H.C. Cheng, R. de Lemos, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2009.

[CMS09] E. Cakar and C. Müller-Schloer. Self-organising interaction patterns of homogeneous and heterogeneous multi-agent populations. In *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 165–174, San Francisco, California, USA, September 2009. IEEE Computer Science.

[Com06] A. Computing. An architectural blueprint for autonomic computing. *White paper*, 36:34, 2006.

[CPGS09] S.W. Cheng, V. Poladian, D. Garlan, and B. Schmerl. Improving architecture-based self-adaptation through resource prediction. In B.H.C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 71–88. Springer-Verlang, 2009.

[CPZ11] G. Cabri, M. Puviani, and F. Zambonelli. Towards ataxonomy of adaptive agent-based collaboration patterns for autonomic service ensembles. In *2011 Annual Conference on Collaborative Technologies and Systems*, pages 306–315, Philadelphia (USA), May 2011.

[CSPSO11] C.E. Cuesta, J. Santiago Prez-Sotelo, and S. Ossowski. Self-organising adaptive structures: The shifter experience, 2011.

[CTN07] H.Q. Chong, A.-H. Tan, and G.-W. Ng. Integrated cognitive architectures: a survey. *Artificial Intelligence Review*, 28(2):103–130, 2007.

[DA00] A.K. Dey and G.D. Abowd. Towards a better understanding of context and context-awareness. In *CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness*, pages 304–307, The Hague, The Netherlands, April 2000. Springer-Verlag.

[Dal11] F. Dalpiaz. *Exploiting Contextual and Social Variability for Software Adaptation*. PhD thesis, DISI- University of Trento, 2011.

[Dav11] J.G. Davis. From crowdsourcing to crowdservicing. *Internet Computing, IEEE*, 15(3):92–94, 2011.

[DB11] S. Dustdar and K. Bhattacharya. The social compute unit. *Internet Computing, IEEE*, 15(3):64–69, 2011.

[DCDG05] G. Di Caro, F. Ducatelle, and L.M. Gambardella. AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Transactions on Telecommunications*, 16(5):443–455, 2005.

[DKP03] T.T. Do, M. Kolp, and A. Pirotte. Social patterns for designing multi-agent systems, 2003.

[DMSFH+04] G. Di Marzo Serugendo, N. Foukia, S. Hassas, A. Karageorgos, S.K. Mostéfaoui, O.F. Rana, M. Ulieru, P. Valckenaers, and C.V. Aart. Self-organisation: Paradigms and applications. In S. A. Brueckner, G. Di Marzo Serugendo, and D. Hales, editors, *Engineering Self-Organising Systems*, volume 3910 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2004.

[DWH06] T. De Wolf and T. Holvoet. Design patterns for decentralised coordination in self-organising emergent systems. In S. A. Brueckner, G. Di Marzo Serugendo, and D. Hales, editors, *Engineering Self-Organising Systems*, volume 3901 of *Lecture Notes in Computer Science*, pages 28–49. Springer, 2006.

[EM10] N. Esfahani and S. Malek. On the role of architectural styles in improving the adaptation support of middleware platforms. In *ECSA'10 Proceedings of the 4th European conference on Software architecture*, pages 433–440, Copenhagen, Denmark, August 2010. Springer-Verlang.

[ERAS+01] M. Esteva, J.A. Rodriguez-Aguilar, C. Sierra, P. Garcia, and J. Arcos. On the formal specification of electronic institutions. In F. Dignum and C. Sierra, editors, *Agent mediated electronic commerce*, volume 1991 of *Lecture Notes in Computer Science*, pages 126–147. Springer, 2001.

[FGG+06] P. Feiler, R.P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, L. Northrop, D. Schmidt, K. Sullivan, et al. *Ultra-large-scale systems: The software challenge of the future*. Carnegie Mellon University, Software Engineering Institute, 2006.

[FMPT01] A. Fuxman, J. Mylopoulos, M. Pistore, and P. Traverso. Model checking early requirements specifications in Tropos. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, pages 174–181, Washington, DC, USA, 2001. IEEE Computer Society.

[FSJ98] P. Faratin, C. Sierra, and N.R. Jennings. Negotiation decision functions for autonomous agents. *International Journal of Robotics and Autonomous Systems*, 24(3-4):159–182, 1998.

[GAKT05] S. Graupner, A. Andrzejak, V. Kotov, and H. Trinks. Adaptive service placement algorithms for autonomous service networks. In S.A. Brueckner, G. Di Marzo Serugendo, A. Karageorgos, and R. Nagpal, editors, *Engineering Self-Organising Systems*, volume 3464 of *Lecture Notes in Computer Science*, pages 280–297. Springer, 2005.

[Gel09] E. Gelenbe. Steps towards self-aware networks. *Communications of the ACM*, 52(7):66–75, 2009.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, Reading (MA), 1995.

[GHK+10] H. Gomaa, K. Hashimoto, M. Kim, S. Malek, and D.A. Menasc. Software adaptation patterns for service-oriented architectures. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 462–469, Sierre, Switzerland, March 2010. ACM.

[GLPT12] E. Gjondrekaj, M. Loreti, R. Pugliese, and F. Tiezzi. Modeling adaptation with a tuple-based coordination language. In *2012 ACM Symposium on Applied Computing*, page to appear, Riva Del Garda (I), March 2012.

[GPS10] C. Ghezzi, M. Pradella, and G. Salvaneschi. Programming language support to context-aware adaptation: a case-study with erlang. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 59–68, Cape Town, South Africa, May 2010. ACM.

[GPS11] C. Ghezzi, M. Pradella, and G. Salvaneschi. An evaluation of the adaptation capabilities in programming languages. In *Proceedings of 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 50–59, Waikiki, Honolulu, Hawaii, May 2011. ACM.

[GRB+05] V. Grimm, E. Revilla, U. Berger, F. Jeltsch, W.M. Mooij, S.F. Railsback, H.H. Thulke, J. Weiner, T. Wiegand, and D.L. DeAngelis. Pattern-oriented modeling of agent-based complex systems: lessons from ecology. *Science*, 310(5750):987–991, 2005.

[Hay08] B. Hayes. Cloud computing. *Communications of the ACM*, 51(7):9–11, 2008.

[HG11] N. Hocine and A. Gouaich. A survey of agent programming and adaptive serious games, 2011.

[HKC+06] 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.

[HL05] B. Horling and V. Lesser. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(4):281–316, 2005.

[Hoh07] G. Hohpe. Soa patterns–new insights or recycled knowledge?, 2007.

[HTE97] A.F. Harmsen, Universiteit Twente, and M. Ernst. *Situational method engineering*. Moret E. & Young Management Consultants, 1997.

[HW11] Matthias Hlzl and Martin Wirsing. Towards a system model for ensembles. In *Festschrift in honor of Carolyn Talcott*, volume 7000 of *LNCS*. Springer, 2011.

[HWB+11] N. Hoch, B. Werther, H. P. Bensler, N. Masuch, M. Ltzenberger, A. Heler, S. Albayrak, and R. Y. Siegwart. A user-centric approach for efficient daily mobility planning in e-vehicle infrastructure networks. In G. Meyer and J. Valldorf, editors, *Advanced Microsystems for Automotive Applications 2011*, VDI-Buch, pages 185–198. Springer-Verlag, 2011.

[HWHJ09] R. Haesevoets, D. Weyns, T. Holvoet, and W. Joosen. A formal model for self-adaptive and self-healing organizations. In *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on*, pages 116–125, Vancouver, BC, Canada, May 2009. IEEE Computer Society.

[JDHS05] W. Jiao, J. Debenham, and B. Henderson-Sellers. Organizational models and interaction patterns for use in the analysis and design of multi-agent systems. *Web Intelligence and Agent Systems*, 3(2):67–83, 2005.

[JFL+01] N.R. Jennings, P. Faratin, A.R. Lomuscio, S. Parsons, MJ Wooldridge, and C. Sierra. Automated negotiation: prospects, methods and challenges. *Group Decision and Negotiation*, 10(2):199–215, 2001.

[JRL09] M.A. Janssen, N.P. Radtke, and A. Lee. Pattern-oriented modeling of commons dilemma experiments. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 17(6):508 – 523, 2009.

[KB00] M.J.B. Krieger and J.B. Billeter. The call of duty: Self-organised task allocation in a population of up to twelve mobile robots. *Robotics and Autonomous Systems*, 30(1):65–84, 2000.

[KBD08] H. Kasinger, B. Bauer, and J. Denzinger. The meaning of semiochemicals to the design of self-organizing systems. In *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 139–148, Isola di San Servolo (Venice), Italy, October 2008. IEEE Computer Society.

[KBD09] H. Kasinger, B. Bauer, and J. Denzinger. Design pattern for self-organizing emergent systems based on digital infochemicals. In *Engineering of Autonomic and Autonomous Systems, IEEE International Workshop on*, pages 45–55, Durham, UK, April 2009. IEEE Computer Society.

[KBM98] D. Kortenkamp, R.P. Bonasso, and R. Murphy. *Artificial intelligence and mobile robots: case studies of successful robot systems*. MIT Press Cambridge, MA, USA, 1998.

[KC03] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[KGGH03] J. Koehler, C. Giblin, D. Gantenbein, and R. Hauser. On autonomic computing architectures. *Research Report (Computer Science) RZ*, 3487, 2003.

[KHKS09] A. Khalid, M.A. Haye, M.J. Khan, and Shamail S. Survey of frameworks, architectures and techniques in autonomic computing. In *ICAS '09 Proceedings of the 2009 Fifth International Conference on Autonomic and Autonomous Systems*, pages 220–225, Valencia, Spain, April 2009. IEEE Computer Socity.

[KP98] N. Karacapilidis and D. Papadias. Hermes: supporting argumentative discourse in multi-agent decision making. In *In Proceedings of the AAAI-98*, pages 827–832, Madison, Wisconsin, USA, September 1998.

[KT11] J. Kesäniemi and V. Terziyan. Agent-environment interaction in mas-introduction and survey. In *Multi-Agent Systems Modeling, Interactions, Simulations and Case Studies*, pages 203–226. Springer Verlag, 2011.

[LB91] A.B. Loyall and J. Bates. Hap – a reactive, adaptive architecture for agents. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991.

[LKB00] A. Ledeczi, G. Karsai, and T. Bapty. Synthesis of self-adaptive software. In *Aerospace Conference Proceedings*, pages 501–507, Big Sky Montana, March 2000. IEEE Computer Science.

[LKMU08] E. Letier, J. Kramer, J. Magee, and S. Uchitel. Deriving event-based transition systems from goal-oriented requirements models. *Automated Software Engineering*, 15(2):175–206, June 2008. Kluwer Academic Publishers, Hingham, MA, USA.

[LNGG11] M. Luckey, B. Nagel, C. Gerth, and Engels G. Adapt cases: Extending use cases for adaptive systems. In *Proceedings of 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 30–39, Waikiki, Honolulu, Hawaii, May 2011. ACM.

[McG03] J.P. McGinnis. Transformations of dynamic interaction protocols in multi-agent systems, 2003.

[MHH11] Z. Maamar, H. Hacid, and M.N. Huhns. Why web services need social networks. *Internet Computing, IEEE*, 15(2):90–94, 2011.

[MHLL08] H. Mei, G. Huang, L. Lan, and J.G. Li. A software architecture centric self-adaptation approach for internetware. *Science in China Series F: Information Sciences*, 51(6):722–742, 2008.

[MK06] J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. John Wiley and Sons, second edition, April 2006.

[MMTZ06] M. Mamei, R. Menezes, R. Tolksdorf, and F. Zambonelli. Case studies for self-organization in computer science. *Journal of Systems Architecture*, 52(8-9):443–460, 2006.

[MPR07] N. Maudet, S. Parsons, and I. Rahwan. Argumentation in multi-agent systems: Context and recent developments. In I. Rahwan and P. Moraitis, editors, *Argumentation in Multi-Agent Systems*, volume 4766 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlang, 2007.

[MPS08] H. Müller, M. Pezzè, and M. Shaw. Visibility of control in adaptive systems. In *Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems*, pages 23–26, Leipzig, Germany, May 2008. ACM Press.

[MSS+10] M. Morandini, L. Sabatucci, A. Siena, J. Mylopoulos, L. Penserini, A. Perini, and A. Susi. On the use of the goal-oriented paradigm for system design and law compliance reasoning. In *iStar 2010–Proceedings of the 4 th International i* Workshop*, page 71, Hammamet, Tunisia, June 2010.

[MT04] R. Menezes and R. Tolksdorf. Adaptiveness in linda-based coordination models. In G. Di Marzo Serugendo, A. Karageorgos, O.F. Rana, and F. Zambonelli, editors, *Engineering Self-Organising Systems*, volume 2977 of *Lecture Notes in Computer Science*, pages 212–232. Springer, 2004.

[MZ09] Marco Mamei and Franco Zambonelli. Programming pervasive and mobile computing applications: The tota approach. *ACM Transactions on Software Engineering and Methodologies*, 18(4), 2009.

[OGT+99] P. Oreizy, M.M. Gorlick, R.N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D.S. Rosenblum, A.L. Wolf, and E.L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, 1999.

[OV02] E. Ogston and S. Vassiliadis. A peer-to-peer agent auction. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 151–159, Bologna, Italy, July 2002. ACM Press.

[PG03] M.P. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Communications of the ACM*, 46(10):25–28, 2003.

[PJ96] S. Parsons and N.R. Jennings. Negotiation through argumentation - a preliminary report. In *Proceedings of the 2nd International Conference on Multi Agent Systems*, pages 267–274, Melbourne, VIC, Australia, July 1996. ACM Press.

[Puv11] M. Puviani. Adaptation in the robotics case study: Early simulation experiences. ASCENS Project Technical Report No. 02, October 2011.

[RC10] A.J. Ramirez and B.H.C. Cheng. 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, Cape Town, South Africa, May 2010. ACM.

[RCMB10] A.J. Ramirez, B.H.C. Cheng, P.K. McKinley, and B.E. Beckmann. Automatically generating adaptive logic to balance non-functional tradeoffs during reconfiguration. In *Proceeding of the 7th international conference on Autonomic computing*, pages 225–234, Washington, DC, USA, June 2010. ACM.

[RLS+11] K. Rasch, F. Li, S. Sehic, R. Ayani, and S. Dustdar. Context-driven personalized service discovery in pervasive environments. *World Wide Web*, 14(4):295–319, 2011.

[RR01] J. Ralyté and C. Rolland. An assembly process model for method engineering. In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, *Advanced information systems engineering*, volume 2068 of *Lecture Notes in Computer Science*, pages 267–283. Springer, 2001.

[RSZF09] J.F. Roberts, T.S. Stirling, J.C. Zufferey, and D. Floreano. 2.5 d infrared range and bearing system for collective robotics. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 3659–3664, St. Louis, MO, USA, October 2009. IEEE Computer Society.

[SFJ99] C. Sierra, P. Faratin, and N. Jennings. A service-oriented negotiation model between autonomous agents. *in Proceedings of Collaboration between Human and Artificial Societies*, pages 201–219, 1999.

[SLT+03] E. Sahin, T.H. Labella, V. Trianni, J.L. Deneubourg, P. Rasse, D. Floreano, L. Gambardella, F. Mondada, S. Nolfi, and M. Dorigo. SWARM-BOT: Pattern formation in a swarm of self-assembling mobile robots. In *Systems, Man and Cybernetics, 2002 IEEE International Conference on*, pages 145–150, Hammamet, Tunisia, October 2003. IEEE.

[Smi06] R.G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *Computers, IEEE Transactions on*, C-29(12):1104–1113, 2006.

[SS05] J. Sabater and C. Sierra. Review on computational trust and reputation models. *Artificial Intelligence Review*, 24(1):33–60, 2005.

[ST09] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):14, 2009.

[TB99] G. Theraulaz and E. Bonabeau. A brief history of stigmergy. *Artificial life*, 5(2):97–116, 1999.

[TOH99] Y. Tahara, A. Ohsuga, and S. Honiden. Agent system development method based on agent patterns. In *Proceedings of the 21st international conference on Software engineering*, pages 356–367, Los Angeles, CA, USA, May 1999. ACM.

[TPYZ09] S. Tang, X. Peng, Y. Yu, and W. Zhao. Goal-directed modeling of self-adaptive software architecture. In T. Halpin, J. Krogstie, E. Nurcan, S.and Proper, R. Schmidt, P. Soffer, and R. Ukor, editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 29 of *Lecture Notes in Business Information Processing*, pages 313–325. Springer, 2009.

[TUV07] TUV. D1.2 Discovering Service-Interaction Patterns-Methods and Mining Algorithms . Technical report, Vienna University of Technology, Austria, 2007.

[VBH+07] J. Vokřínek, J. Bíba, J. Hodík, J. Vybíhal, and M. Pěchouček. Competitive contract net protocol. In J. Van Leeuwen, G.F. Italiano, W. Van Der Hoek, C. Meinel, H. Sack, and F. Plasik, editors, *SOFSEM 2007: Theory and Practice of Computer Science*, volume 4362 of *Lecture Notes in Computer Science*, pages 656–668. Springer, 2007.

[VG10] T. Vogel and H. Giese. Adaptation and abstract runtime models. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 39–48, Cape Town, South Africa, May 2010. ACM.

[VHGN11] Emil Vassev, Mike Hinchey, Benoit Gaudin, and Paddy Nixon. Requirements and initial model for knowlang: a language for knowledge representation in autonomic service-component ensembles. In *Fourth International C* Conference on Computer Science & Software Engineering*, pages 35–42. ACM, 2011.

[vLDDD91] A. van Lamsweerde, A. Dardenne, B. Delcourt, and F. Dubisy. The KAOS project: Knowledge acquisition in automated specification of software. In *Proceedings of the AAAI Spring Symposium Series*, pages 59–62. Stanford University, American Association for Artificial Intelligence, March 1991.

[VM09] B. Varghese and G. McKee. Applying autonomic computing concepts to parallel computing using intelligent agents. *World Academy of Science, Engineering and Technology*, 55:366–370, 2009.

[VRHR11] P. Van Roy, S. Haridi, and A. Reinefeld. Designing robust and adaptive distributed systems with weakly interacting feedback structures. Technical report, ICTEAM Institute, Universit catholique de Louvain, 2011.

[VS03] J. Vázquez-Salceda. The role of Norms and Electronic Institutions in Multi-Agent Systems applied to complex domains. The HARMONIA framework. *AI Communications*, 16(3):209–212, 2003.

[VWMA11] P. Vromant, D. Weyns, S. Malek, and J. Andersson. On interacting control loops in self-adaptive systems. In *Proceedings of 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Waikiki, Honolulu, HI, USA, May 2011. ACM.

[WDNG+06] M. Wirsing, R. De Nicola, S. Gilmore, M.M. Hölzl, R. Lucchi, M. Tribastone, and G. Zavattaro. Sensoria process calculi for service-oriented computing. In *2nd International Symposium on Trustworthy Global Computing*, pages 30–50, Lucca, Italy, November 2006. Springer.

[WG09] D. Weyns and M. Georgeff. Self-adaptation using multiagent systems. *Software, IEEE*, 27(1):86–91, 2009.

[WH07] D. Weyns and T. Holvoet. An architectural strategy for self-adapting systems. In *Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, page 3, Minnesota, USA, May 2007. IEEE Computer Society.

[WMA10a] D. Weyns, S. Malek, and J. Andersson. FORMS: a formal reference model for self-adaptation. In *Proceeding of the 7th international conference on Autonomic computing*, pages 205–214, Washington, DC, USA, June 2010. ACM.

[WMA10b] D. Weyns, S. Malek, and J. Andersson. On decentralized self-adaptation: lessons from the trenches and challenges for the future. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 84–93, Cape Town, South Africa, May 2010. ACM.

[WMdLA10] D. Weyns, S. Malek, R. de Lemos, and J Andersson. Self-organizing architectures, first international workshop, soar 2009, cambridge, uk, september 14, 2009, revised selected and invited papers. In D. Weyns, S. Malek, R. de Lemos, and J Andersson, editors, *SOAR*, volume 6090 of *Lecture Notes in Computer Science*. Springer, 2010.

[WWW98] P.R. Wurman, M.P. Wellman, and W.E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In *Proceedings of the second international conference on Autonomous agents*, pages 301–308, St. Paul, Minnepolis, USA, May 1998. ACM Press.

[ZBC+11] F. Zambonelli, N. Bicocchi, G. Cabri, L. Leonardi, and M. Puviani. On self-adaptation, self-expression, and self-awareness, in autonomic service component ensembles. In *AWARENESS Workshop at the 5th IEEE International Conference on Self-adaptive and Self-organizing Systems*, Ann Arbor (MC), 2011.

[ZF+97] L. Zhang, S. Floyd, et al. Adaptive web caching. In *In Proceedings of the NLANR Web Cache Workshop*, pages 7–9, Boulder, Colorado, USA, June 1997.

[ZJW03] F. Zambonelli, N.R. Jennings, and M. Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, 2003.