SOTA: Towards a General Model for Self-Adaptive Systems

Dhaminda B. Abeywickrama, Nicola Bicocchi, Franco Zambonelli
Department of Science and Methods for Engineering
University of Modena and Reggio Emilia, Italy
Email: {dhaminda.abeywickrama, nicola.bicocchi, franco.zambonelli}@unimore.it

***Abstract*—The increasing complexity and dynamics in which software systems are deployed call for solutions to make such systems autonomic, i.e., capable of dynamically self-adapting their behavior in response to changing situations. To this end, proper models and software engineering tools are required to be available to support the design and development of autonomic systems. In this paper, we introduce a new general model, SOTA, for modeling the adaptation requirements. SOTA, by bringing together the lessons of goal-oriented modeling and of context-aware system modeling, has the potentials for tackling some key issues in the design and development of complex self-adaptive software systems. In particular, SOTA enables: early verification of requirements, identification of knowledge requirements for self-adaptation, and identification of the most suitable self-adaptive patterns.**

***Keywords*-self-adaptive systems; goal-oriented requirements engineering; architectural patterns; model checking**

I. Introduction

In the past few years, several research works have been devoted to identify models, languages, and tools, to support the development of software systems that are at the same time predictable in behavior and autonomically adaptive to respond to contingencies [1], [2]. In particular, a key open issue is the identification of a general modeling framework that can tackle many complex issues of self-adaptive software engineering. These include: proper analysis and verification of functional and non-functional requirements; analysis and identification of the knowledge requirements (i.e., of which knowledge must be made available to support self-adaptive behavior); and identification of the most suitable architectural design patterns to achieve self-adaptation.

To tackle this issue, we have adopted a sort of "black-box" approach to adaptation in which, abstracting from the actual mechanisms of adaptation, 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 SOTA ("State Of The Affairs"), a robust conceptual framework that, grounding on the lessons of goal-oriented requirements engineering [3] and multidimensional context modeling [4], can act as an effective conceptual support to self-adaptive software development.

In particular, the key idea in SOTA is to perceive a self-adaptive software system as a sort of complex dynamic system immersed in a virtual n-dimensional phase space, each dimension being associated to an internal or environmental parameter of interest for the execution of the system. The adaptive execution of the system can then be modeled in terms of movements in such space. Functional requirements (i.e., goals) are associated to areas of the phase space the system has to reach, non-functional requirements are associated to the trajectory the system should try to walk through, self-adaptation is associated to the capability of the system to re-join proper trajectories when moved away from it.

Our research and development of the SOTA model is still at a rather preliminary stage. Yet, as we show in this paper, SOTA (described in Section II) promises to exhibit several advantages and proposes itself as a general-purpose and effective framework. In particular: SOTA can be used to early assess self-adaptation requirements via model-checking techniques (as described in Section III); it can be used as a tool to support the process of identifying which knowledge must be made available to the system and its components, to support adaptivity (Section IV); and it can effectively support designers in the most appropriate internal software structures and processes to adopt in self-adaptive system design (as discussed in Section V).

II. The SOTA Model

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, requirements modeling is a key point to start framing the concept of adaptation.

A. Motivations and Rationale

Traditionally, in the software engineering area, the requirements can be divided into two categories: functional requirements (what the system should do) and non-functional requirements (how the system should act in achieving its functional requirements, e.g., in terms of system performances, quality of service, etc.).

In the area of adaptive systems, and more in general of open-ended systems immersed in dynamic environments

1524-4547/12 $26.00 © 2012 IEEE
DOI 10.1109/WETICE.2012.48

both functional and non-functional requirements are better expressed in terms of "goals" [3]. A goal, in general terms, is a "state of the affairs" that an entity aims to achieve.

The idea of goal-oriented modeling of requirements naturally matches goal-oriented and intentional entities (e.g., humans, organizations and multi-agent systems). However, since self-adaptation is naturally perceivable as an "intentional" quality, goal-oriented modeling appears the natural choice for self-adaptive systems, since it matches the "observable" intentionality of a system that takes actions aimed at adapting its behavior.

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.

As for the "state of the affairs" (from which the SOTA acronym derives), this 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 system.

Against this background, SOTA builds on the most assessed approaches to goal-oriented requirements engineering [3]. For modeling the adaptation dimension, SOTA integrates and extends recent approaches on multidimensional modeling of context such as the Hyperspace Analogue to Context (HAC) approach [4]. In particular, such generalization and extensions try to account for the general needs of dynamic self-adaptive systems and components.

B. The SOTA Space

SOTA assumes that the current "state of the affairs" $S(t)$ at time t, of a specific entity e (let it be an individual component or an ensemble) can be described as a tuple of n s_i values, each representing a specific aspect of the current situation:

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

As the entity executes, 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^e as a movement in a virtual n-dimensional space $\mathbf{S}$ (see Fig.1):

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

Or, according 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.

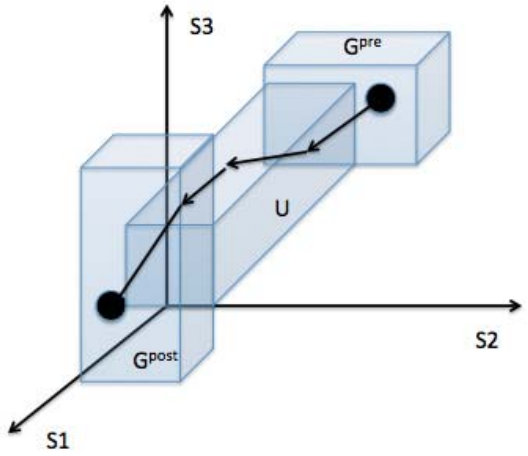

Figure 1. The trajectory of an entity in the SOTA space, starting from a goal precondition and trying to reach the postcondition while moving in the area specified by the utility.

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.,

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

A transition can be endogenous, i.e., induced by actions within the system itself, or exogenous, i.e., induced by external sources. The existence of exogenous 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 factors requiring adaptation.

C. Goals and Utilities

A *goal* by definition is the eventual achievement of a given state of the affairs. Therefore, in very general terms, a specific goal G_i for the entity e can be represented as a specific point, or more generally as a specific area, in such space. That is:

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

Getting more specific, a goal G_i of an entity e may not necessarily be always active. Rather, it can be the case that a goal of an entity will only get activated when specific conditions occur. In these cases, it is useful to characterize a goal in terms of a precondition G_i^{pre} and a postcondition G_i^{post}, to express when the goal has to activate and what the achievement of the goal implies. 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 Fig.1). Clearly, a goal with no precondition is like a goal whose precondition coincides with the whole space, and it is intended as a goal that is always active.

As goals represent the eventual state of the affairs that a system or component has to achieve, they can be considered functional requirements. However, in many cases, a system should try to reach its goals by adhering to specific constraints on how such a 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 sometimes an entity should try (or be constrained) to reach a goal by having its trajectory be confined within a specific area (see Fig.1). We call these sorts of constraints on the execution path that a system/entity should try to respect as *utilities*. This is to reflect a nature that is similar to that of non-functional requirements.

As goals, a utility U_i can be typically expressed as a subspace in $\mathbf{S}$, and can be either a general one for a system/entity (the system/entity must always respect the utility during its execution) or one specifically associated to a specific goal G_i (the system/entity should respect the utility while trying to achieve the goal). For the 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 constraint (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 postcondition, and possibly associated goal-specific utilities) and the global utilities for such systems, that is, the sets:

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

Of course, during the identification of goals and utilities, it is already possible to associate goals and utilities locally to specific components of the system as well as globally, to the system as a whole. Thus, the above sets can be possibly further refined by partitioning them among local and global ones.

The discussion in the following sections details how SOTA represents a useful tool for the engineering of self-adaptive systems.

III. Model Checking SOTA Requirements

In this section we discuss how it is possible to adopt SOTA as an effective tool to perform an early, goal-level, *model checking analysis* [5] for adaptive systems. The specific advantages of this approach are related to exploiting the goal-oriented modeling of SOTA where 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. Also, the approach can exploit event-based systems for automating the formal analysis of software architecture specifications, and for supporting software architecture design and program verification and testing. Our approach is based on the formal verification, validation and simulation techniques provided by the Labeled Transition System Analyzer (LTSA) [5].

As described in detail in [6], model checking has been specifically applied to: *(i)* validate whether a set of required preconditions and postconditions forms a complete operationalization of a single goal (i.e., single goal operationalization); *(ii)* check the satisfaction of higher-level goals, which allows the engineer not to confine verification to a single goal or utility but a portion of the goal graph (i.e., multiple goal operationalization); *(iii)* detect any inconsistencies and implicit requirements as deadlocks; *(iv)* perform animation of the goal-oriented models using the standard animation and simulation features of the LTSA. The counterexample traces generated and the deadlocks detected in the model checking process are used to iteratively refine and improve the SOTA requirements model, thus deriving a specification that is correct.

A. Validate Single Goal Operationalization

In single goal operationalization, we validate whether a set of preconditions, postconditions and/or a goal-associated utility forms a complete operationalization of a requirement R. A goal G_i can be characterized by a precondition G_i^{pre} and a postcondition G_i^{post}, and goal G_i can be associated to a utility U_i that needs to be respected while trying to achieve G_i.

If the operationalization is not complete the LTSA model checker generates a counterexample trace. For example, this can be the omitting of a required precondition during the operationalization process.

B. Validate Higher-Level Goal Satisfaction

In addition to validating a single goal operationalization we perform model checking to check the satisfaction of higher-level goals in the SOTA operational model. This allows the engineer not to confine verification to a single

goal or utility but a portion of the goal graph. In SOTA terms, this is checking the requirements of the set of goals G and utilities U:

$$\mathbf{G} = \{G_1, G_2, \ldots, G_n\}, |\mathbf{G}| > 1$$

$$\mathbf{U} = \{U_1, U_2, \ldots, U_n^e\}, |\mathbf{U}| > 1$$

C. Detect Inconsistencies and Implicit Requirements

A typical problem that can occur in goal-oriented modeling is that an *inconsistency* or an *implicit requirement* can result in a deadlock in the specification. An inconsistency in the specification can occur for several reasons. These can be *(i)* if the postcondition of a goal does not imply its precondition then the system might be in a state where the postcondition G_i^{post} is true but the precondition G_i^{pre} is not. So the goal needs to be satisfied but it is not, leading to an inconsistency; *(ii)* if the operational model is derived from conflicting goals. Therefore, it is important to detect inconsistencies in the SOTA operation model.

As for the inconsistencies that arise from conflicting goals, let us consider two service components SC_1 and SC_2 that are to be composed into an ensemble SCE. First, assume that SC_1 and SC_2 have two shared goals G_i and G_j, which share the same n-dimensional SOTA space S. The preconditions of the two goals overlap but the postconditions do not overlap. That is:

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

Therefore, both these goals can be activated and pursued at the same time in two paths in the SOTA space S, and this should not be the case. Second, assume that SC_1 and SC_2 have two goals G_i and G_j and the goals' preconditions and postconditions both overlap. That is:

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

Therefore, both these goals can be activated and pursued at the same time in the same direction of the SOTA space S. We perform LTSA model checking to detect such inconsistencies that arise from conflicting goals as deadlocks in the specification.

On the other hand, *implicit requirements*, occur due to interactions between requirements on different goals. For example, a postcondition of a goal G_i^{post} may implicitly prevent another goal being applied even if all the preconditions for the second goal G_j^{pre} are true. This is because the actual condition G_i^{post} in which the goal is allowed to occur is stronger than the preconditions G_j^{pre} of that goal. 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 eventually derive a robust specification.

D. Goal-based Animation

The goal-oriented requirements models of SOTA can be *animated* using the standard animation and simulation features of the LTSA tool. The goal-based models can be explored interactively with stakeholders and error traces generated during formal analysis can be replayed, increasing the confidence in the validity of the goals and utilities.

IV. 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 (i.e., hardware, software, environmental features). Thus, identifying which are the relevant dimensions around which to model the SOTA space is a necessary activity towards the building of a self-adaptive system.

However, when talking about self-adaptive systems, such identification also directly relates to identify *(i)* which knowledge (i.e., which dimensions) must be made available to components to enable self-adaptation; *(ii)* which kind of sensors (either physical or virtual) have to be available from which components to gather the necessary knowledge, and *(iii)* which kinds of *metrics* can be applied to such knowledge.

A. Identification

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 dimension S_i is not relevant for either a specific goal or utility, then $A_i \equiv \mathbf{S}_i$.

Indeed, in many practical cases, goals and utilities involve only a limited fraction of the state 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. 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. That is, it drives the requirements for which knowledge has to be acquired, modeled, and made available to services.

In addition, one should also account for specific contingent situations of 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 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.

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 characteristics of the domain can induce additional constraints. 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.

B. Sensors and Virtualization

Each dimension of the SOTA space implies sensors. However, this is not an issue *per se*: a number of different sensors are available to measure the most diverse features. The problem, instead, lies in providing services with an appropriate view of what's happening, i.e., leveraging the low-level perspective of the actual sensors into that of a "virtual sensor", capable of providing an appropriate view representation of the values in that dimension. For example, accelerometer sensors report movements in terms of numeric time series, whereas within the SOTA space it would better consider movements as represented in terms of values such as "looking ahead", "sleeping", "looking rear".

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, while the most obvious choice for producing such virtual information are smart phone accelerometers, one could also adopt image processing techniques or even couple image processing with accelerometers to fuse both the contributions.

In general, virtual sensors are useful for: *(i)* grouping a number of physical sensors for the sake of fault tolerance; *(ii)* converting sensor readings into relevant information; and *(iii)* grouping different physical sensors allowing multimodal 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.

C. Metrification

The above process of virtualization implicitly carries the definition of the granularity of transition. That is, a transition for a component e 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 dimensions are identified, it may be necessary to associate a "metric" to the values in that space, enabling systems assessing their direction within the state space.

In many cases dimensions can be expressed as metric spaces of numerical values. In case of nominal values, instead, the issue of defining a proper metric arises. 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.

The above considerations about the modeling of the SOTA space eventually ends up in identifying in a quite accurate knowledge requirements definition. In general terms, the modeling of the SOTA space, helps clearly separating the phases related to the modeling of the system component to the one related to the modeling of the knowledge that they have to manage.

V. Self-Adaptation Patterns

The SOTA modeling approach is very useful for understanding and modeling the functional and adaptation requirements, and for checking the correctness of the specification. However, when a designer considers the actual "white box" design of the system, it is important to identify which architectural schemes need to be chosen for the individual components and the service component ensembles.

To this end, in previous work [7], we defined a taxonomy of architectural patterns for adaptive components and ensemble of components. This taxonomy has the twofold goal of enabling reuse of existing experiences in the area and providing useful suggestions to a designer on selecting the most suitable patterns to support adaptability. Here we sketch how to redefine such taxonomy in SOTA terms, to make it more useful and clean.

At the center of our taxonomy is the idea that self-adaptivity requires the presence (explicit or implicit) of a *feedback loop* or control loop. A feedback loop is the part of the system that allows for feedback and self-correction towards goals' achievement, i.e., self-adjusting behavior in response to the system's changes. It provides a generic mechanism for adaptation where it provides means for inspecting and analyzing the system at the service component or ensemble level and for reacting accordingly. In SOTA, we can use feedback loops to describe how goals and utilities are expressed in the adaptive patterns.

SOTA defines several architectural design patterns at the service component and ensemble levels, as outlined next (refer to [7] for further details).

The *primitive service component* is a non-adaptive pattern that is simply capable of providing functions. The *reactive service component* modifies its behavior, reacting to external events from the environment. It has no feedback loops present, so the adaptation is a result of its interactions with the environment. The reactive service component has the capability of perceiving the SOTA space and applying actions to modify its position in the space in the medium term (i.e., it has utilities). But it has no means to aim in the

long term, so it does not have goals. That is:

$$\mathbf{G} = \emptyset, \mathbf{U} = U_1, U_2, \ldots, U_n$$

An example of a component that uses this pattern can be found in the Hap architecture [8].

The *goal-oriented service component* has internal control loops and the adaptation is explicit. It not only reacts to external events but also actively tries to adapt its goal or utility-oriented behavior. In addition to having utilities this component can apply actions in the SOTA space in the long term (i.e., it has goals). That is:

$$\mathbf{G} = G_1, G_2, \ldots, G_m, \mathbf{U} = U_1, U_2, \ldots, U_n$$

An example of this pattern can be found in Jadex [9], which exploits the BDI model to support cognitive agents.

The *autonomic service component* (ASC) is characterized by an explicit external feedback loop. This pattern is composed of a controller service component (CSC) and an autonomic control manager (ACM) at the end of the loop to monitor and direct behavior. The services, and goals and utilities of the CSC (if any) are tightly coupled with the ACM. Like the goal-oriented service component this pattern has goals and utilities. That is:

$$\mathbf{G_{ASC}} = \mathbf{G_{CSC}} \cap \mathbf{G_{ACM}}; \mathbf{U_{ASC}} = \mathbf{U_{CSC}} \cap \mathbf{U_{ACM}}$$

The autonomic computing paradigm called MAPE-K [1] is an example of the use of this pattern.

Adaptation patterns defined for the ensemble level are categorized based on features such as the implicit or explicit nature of the feedback loops and the pattern's goals. Three key pattern categories are: *(i) environment mediated*: ensembles in which the adaptation activities are not explicitly designed but emerge from the interactions of the components and the shared environment (e.g., pheromone based [10]); *(ii) negotiation / competition*: ensembles in which the adaptive behavior is explicitly designed using interaction patterns between components (e.g., choreographies or negotiations [11]); (iii) *norm based*: ensembles in which there is a set of components with a global control loop over the ensembles to control and direct the overall behavior (e.g., coordinated systems and electronic institutions [12]).

VI. Conclusions and Future Work

Our research on the SOTA model is still at an early stage, and we are currently working towards better assessing its potentials and to make it a practical and usable tool. In particular, in the context of the ASCENS ("Autonomic Service Component Ensembles") European Project, we intend to put the SOTA model at work in the experimental prototyping of software systems for supporting e-mobility. In addition, we intend to proceed with the framing and cataloguing of relevant self-adaptive patterns, accurately model them in SOTA terms, and define guidelines and support to assist designers in their choices.

Acknowledgment

This work is supported by the ASCENS project (EU FP7-FET, Contract No. 257414).

References

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

[2] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 4, no. 2, pp. 1–42, 2009.

[3] 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 *Proceedings of the 4th International i* Workshop (iStar'10)*, Hammamet, Tunisia, June 2010, pp. 71–75.

[4] K. Rasch, F. Li, S. Sehic, R. Ayani, and S. Dustdar, "Context-driven personalized service discovery in pervasive environments," *World Wide Web*, vol. 14, no. 4, pp. 295–319, 2011.

[5] J. Magee and J. Kramer, *Concurrency: State Models and Java Programs*, 2nd ed. John Wiley and Sons, Apr. 2006.

[6] D. B. Abeywickrama and F. Zambonelli, "Model checking goal-oriented requirements for self-adaptive systems," in *Proceedings of the 19th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, Apr. 2012, pp. 33–42.

[7] G. Cabri, M. Puviani, and F. Zambonelli, "Towards a taxonomy of adaptive agent-based collaboration patterns for autonomic service ensembles," in *Proceedings of the 2011 International Conference on Collaboration Technologies and Systems*. IEEE, May 2011, pp. 508–515.

[8] A. B. Loyall and J. Bates, "Hap: A reactive, adaptive architecture for agents," Carnegie Mellon University, Pittsburgh, USA, Tech. Rep., Jun. 1991.

[9] L. Braubach, A. Pokahr, and W. Lamersdorf, "Jadex: A BDI-agent system combining middleware and reasoning," in *Software Agent-Based Applications, Platforms and Development Kits*, R. Unland, M. Calisti, and M. Klusch, Eds. Springer, 2005, pp. 143–168.

[10] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, and T. Urnes, "Design patterns from biology for distributed computing," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 1, no. 1, pp. 26–66, Sep. 2006.

[11] C. Beam and A. Segev, "Automated negotiations: A survey of the state of the art," *Wirtschaftsinformatik*, vol. 39, no. 3, pp. 263–268, 1997.

[12] M. Esteva, J. A. Rodríguez-Aguilar, C. Sierra, P. Garcia, and J. Arcos, "On the formal specification of electronic institutions," in *Agent Mediated Electronic Commerce: The European AgentLink Perspective*. Springer-Verlag, 2001, pp. 126–147.