Integrating Semantic Run-Time Models for Adaptive Software Systems

Francesco Poggi[1], Davide Rossi[1] and Paolo Ciancarini[2]

[1]*Department of Computer Science and Engineering (DISI), University of Bologna, Bologna, Italy*
[2]*University of Bologna (Italy) and Innopolis University (Russia)*
E-mail: Francesco.Poggi@unibo.it; Davide.Rossi@unibo.it; paolo.ciancarini@unibo.it

Received 05 January 2019;
Accepted 03 March 2019

Abstract

Software-intensive systems work in ever-changing environments requiring expensive technical efforts to manage their evolution. In order to mitigate their risks and costs they should dynamically self-adapt to any modification of their environment. MAPE-K (Monitor, Analyze, Plan, Execute – Knowledge) is the basic architectural pattern for building software-intensive self-adaptable systems. In this paper we propose an approach in which all the information about a system and its environment is unified by using Semantic Web technologies into a set of semantic run-time models which enhance the Knowledge in MAPE-K. Ontologies are used to manage the interaction and integration of these models with disparate data sources. The resulting knowledge base is then used to drive adaptation activities exploiting well known languages and notations. We discuss how MAPE-K can be exploited in order to take advantage of ontological representations, along with

Journal of Web Engineering, Vol. 18_1-3, 1–42.
doi: 10.13052/jwe1540-9589.18131

Semantic Web languages and tools, by studying a real-word case study: a legacy system that was not designed to perform automatic adaptation. We discuss merits and limits of our approach based on semantic run-time models both in the context of this specific case study and in a broader scope.

Keywords: Autonomic systems, adaptive software, MAPE-K, Semantic Web, ontology.

1 Introduction

Software systems work in continuously changing environments, where endless modifications in user needs, resource availability, and system faults require expensive technical efforts. In order to mitigate the costs for such efforts, software systems should be able to dynamically self-adapt.

Two main approaches to software adaptation exist: internal or external adaption, referring to where the self-reconfiguration is managed [1].

The former consists in using specialized mechanisms (e.g. exceptions in programming languages, fault-tolerant protocols, etc.) at design time to accommodate run-time variabilities. Although effective in practice, these mechanisms have the drawback to be highly specific to the application and wired into the code. As a consequence, systems that use internal approaches are costly to build, difficult to manage and modify, and usually provide only localized treatments to the changes.

In contrast to the internal approach, many recent works suggest to use external mechanisms [2, 3] and models [4] that implement a closed-loop control pattern to monitor and dynamically adapt at run-time the system's behaviors. In the external approach, the adaptation is realized by an autonomous manager component that resides outside the managed elements. By localizing the concerns of monitoring and resolving problems in a separate software component, external approaches provide an effective engineering solution. The issues related to the development, analysis, maintenance, and extension are simplified since they are

isolated in a separate engine, thus reuse across multiple systems is facilitated, as described in [1]. Moreover, the self-adaptation behaviours of an external manager can be integrated into legacy systems, for which documentation and source code is unavailable.

An architecture-based approach to self-adaptation was first developed in [5], exploiting architectural models to represent complex systems in terms of components, relations, and properties. A reference architecture for these adaptive systems has been proposed in [6], and a general and reusable infrastructure based on the classical control loop in control theory has been described in [7].

More recently, the emerging field of Model Driven Engineering (MDE) [8] called Models@Run.time [4] extended this approach, investigating the use of different models to represent the knowledge required for managing the operations of software systems. The main benefit of Models@Run.time is a unified high-level perspective of the observed systems, together with precise representations of the knowledge needed to drive their adaptation.

Model-based adaptation raises new research challenges that have to be tackled [9]. In this work we mainly focus on two problems. The first challenge is *heterogeneity management:* this is the capability of managing the heterogeneity of knowledge sources and knowledge users. Most systems are made of different programming languages, technologies, architectural styles, and mechanisms to support dynamic adjustments; providing a general framework to manage this variability helps in lowering the level of complexity of the overall system and allows easier dynamic integration of new data sources and changes of policies and rules. Moreover, in order to allow adaptation at the model level, the heterogeneity problem is not limited to the technological and architectural aspects of software systems. Since in this paradigm models are the primary means to understand, interact with, configure, and modify the run-time behavior of software, architectural models must be integrated with other ones (e.g. Monitoring Models, Evaluation Models, Change Models, Execution Models, etc.) to capture the different concerns, information and viewpoints of each specific domain.

We exploit ontologies to solve the problem of heterogeneity management. By definition, an ontology is "the manifestation of a shared understanding of a domain that is agreed between a number of agents, and such agreement facilitates accurate and effective communications of meaning" [10], which in turn leads to other benefits such as interoperability, reuse, and sharing. Ontologies address the heterogeneity management problem providing a shared and unified representation of a complex and heterogeneous domain of interest. In particular, we used the Web Ontology Language (OWL) [11] for all the semantic models that represent system and environment information. This choice has the advantage to offer a standardized and mature set of notations and technologies to represent and perform computations on such models. For example, the default Semantic Web query language (i.e. SPARQL [12]) provides an out-of-the-box solution to the run-time queryable models problem. Moreover, adaptation strategies can be implemented using semantic reasoners (e.g. Pellet[13], FaCT++[14], Hermit[15], etc.) and rule languages (e.g. the Semantic Web Rule Language – SWRL [16]).

The second challenge is *dynamic query management*, that is the capability of creating run-time queryable models. Model-based adaptation approaches should guarantee easy and uniform mechanisms to interrogate the information about the observed systems and their environment, possibly accessing data that pertain to different viewpoints. A unified perspective of the observed system should be provided as up-to-date queryable models, ready to be processed by adaptation engines.

In this paper we propose an approach able to effectively tackle both challenges by defining a reference framework for autonomic systems based on the MAPE-K pattern. This framework is then validated by instantiating it to solve a real world case study. A key point of our proposal is the use of specific ontologies we developed for representing system and environment-related information.

The rest of the paper is organized as follows. In Section 2 we discuss some relevant related works. Section 3 introduces a real case study that we will use to illustrate and then evaluate our approach. Section 4 describes the method we used to develop our semantic models. Section 5 illustrates the architecture of the adaptation engine we propose.

Section 6 provides an example of an adaptation loop execution and shows how system information can be queried at run-time. In Section 7 we test and evaluate the effectiveness of our approach for the case study. Section 8 compares our solution with the main approaches to self-adaptation based on run-time models, and presents recent developments of this research line. Finally, in Section 9 we draw some conclusions and discuss some future extensions of this work.

2 Related Work

Semantic technologies have been used diffusely to support adaptation in the context of services composition [17]. The main idea here is to allow the dynamic selection of cooperating services on the basis of their semantic description.

OWL-S [18] and its ancestor, DAML-S [19], are Semantic Web technologies to describe web services. Both DAML-S and OWL-S had among their design goals the ability to describe web services in a way that would foster their composition on the basis of semantic annotations. Relevant examples of this approach include [20] where DAML-S is used with Petri Nets for simulation, verification, and automated composition of Semantic Web services, [21] where the authors propose a UDDI extension for semantic services matchmaking and [22] where the authors propose a semi-automatic approach, in the form of a recommender for composition designers that uses DAML-S.

Other two interesting works use Semantic Web technologies for self-configuring and self-adapting information networks [23], and design an adaptive framework in a public safety context that supports collaborative work and dynamic reconfiguration at run-time [24], respectively. However, very few works treat specifically the problem of providing tools for building context-aware collaborative applications with dynamic reconfiguration of components at runtime.

In the context of agent-based systems and robotics the use of Semantic Web technologies to represent dynamic models has been explored in [25]. The authors propose a framework in which an OWL

ontology is used to describe the semantics and structure of information processing components. A description based on this ontology is then transformed into Answer Set Programming (ASP) [26], a form of declarative programming that is used to represent the overall logic of the problem to solve; then an ASP solver is used to create a run-time model of information processing.

In [27] the authors describe a framework based on sets of *Live Semantic Annotations* and eco-laws, namely self-organizing system rules. These artifacts are automatically translated to RDF and SPARQL to make use of existing Semantic Web inference tools and enable resource discoveries in opportunistic networks.

Many recent works and standards proposes approaches and technologies based on Semantic Technologies to easily model dynamic and self-reconfiguring applications especially for ubiquitous and pervasive computing. One of the first ontology-based approaches is the Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA) [28]. It is expressed in OWL and includes modular component vocabularies to represent intelligent agents, time, space, events, user profiles, actions, and policies for security and privacy. The authors describe two prototype systems that exploit SOUPA to support knowledge sharing, context reasoning, and dynamic system behaviors at run-time.

Internet of Things (IoTs) researchers are taking into consideration Web of Things (WoTs) to support reactive system engineering. The goal of the WoT is to extend Web services to devices, allowing a Web client to access devices properties, to request the execution of actions or to subscribe to events representing state changes [29]. The related ontology describes how to model physical or virtual sensors and actuators with the main objective of easing the binding with phisical devices reachable through web protocols (REST, CoAP, etc.).

A different objective is pursued by the Semantic Sensor Network (SSN) ontology [30], an Open Geospatial Consortium (OGC)/World Wide Web Consortium (W3C) standard. It is mainly focused on the SOSA (Sensor, Observation, Sample, Actuator) pattern [31], a pattern that can be used to effectively model reactive systems. Therefore it aims at supporting the definition of simple adaptive behaviors that link observations, coming from modeled sensors, with the related reactions,

performed by actuators. These behaviors are represented by RDF subgraphs in a knowledge base and can be activated when observation facts are asserted. In order to link observations to physical or virtual properties, the SOSA pattern is extended with some system-oriented features.

The Semantic Smart Sensor Network (S3N) ontology [32] is another effort that tries to specialize SSN for supporting the modeling of smart sensors. To this end a new class, `s3n:SmartSensor`, has been introduced as a specialization of `ssn:System`. A smart sensor is composed of embedded sensors, microcontrollers and communicating systems. The behavior is expressed by the execution of an algorithm (selected among the existing ones on context basis) by the microcontroller, which can be thought as a specialization of the `ssn:Actuator`, being able to select algorithms from the current context and to change the state of the whole smart sensor. Therefore, the main purpose of S3N is to support smart sensors modeling and not to close the logical gap between sensors and actuators for fully programming reactive systems.

Some recent works are based on the idea of using the aforementioned standards and technologies to develope reactive cloud applications. For instance, in [33] authors present a proposal for an ontology-driven approach that leverages techniques from the Semantic Sensor Web to develope self-adaptive cloud application platforms using the MAPE-K reference model. The main advantage of the described framework is the support for run-time analysis of the heterogeneous monitored values by means of reasoning over ontologies and rules. Another example is FRAMESELF [34], an ontology-based framework designed for the self-configuration of Machine-to-machine (M2M) architectures and communications. The main problem faced in this work is the need to connect thousands of heterogeneous machines that are widely distributed and frequently evolve according to their environment changes. The proposed framework uses the MAPE-K pattern for self-managing the integration and reconfiguration of heterogeneous systems.

Effective cloud application platforms self-management is the main objective presented in [35]. The work uses the MAPE-K paradigm

as reference model for adaptation cycles, and ontologies and rules to represent self-reflective system knowledge and define adaptation policies. In [36] the authors use Semantic Web languages to encode values monitored within cloud application platforms, and integrate semantically-enriched observation streams with static ontological knowledge to adapt systems through run-time reasoning mechanisms. A prototype is presented that utilize Stream Reasoning techniques to perform analysis and failure diagnosis, and suggest adaptation actions.

To the best of our knowledge no other work in the field of self-adaptation supported by the MAPE-K pattern includes an extensive use of Semantic Web languages and tools.

3 Case Study

Our idea to base software adaption on semantic models has been tested in the context of a real world use case in order to assess its effectiveness. Our research group has an ongoing collaboration with CeSIA, the center responsible for the whole IT infrastructure of the University of Bologna (UniBo). UniBo provides ICT services to more than 90.000 students and about 10.000 employees, including both faculty and staff. The IT architecture supporting these services is composed of 160 centers connected by 510 km of optical fibers, and by means of 500 servers (90% virtual), exposing 480 websites with an average of 12.400.000 visitors/year and 137.000.000 page hits/year.

One of the main problems faced by our technical staff is the timely management of this huge system: driven by automatic alerts and users, whose feedbacks are collected by the help desk in approximately 50.000 tickets/year, the staff people use a monitor infrastructure to analyze the system state, identify the root cause of each issue, and activate (human-driven) adaptation plans to solve problems.

Current monitoring is mostly focused on the infrastructure and network level and uses Nagios[1], but a plan to extend it has already been designed and includes Bischeck[2] for inspecting applications and

[1]https://www.nagios.org/. This and all others URLs presented in the paper have been accessed on March 20th, 2019.

[2]http://www.bischeck.org/

processes, Splunk[3] for analyzing machine-generated data and New Relic[4] for application monitoring. More details on this case study can be found in [37].

Since each monitor component provides a partial perspective of the whole system, one of the main issues is to link these fragmented and low level data about the system state to higher level informative elements in architectural views, giving clearer and more complete representations of the domain elements and their relationships. This relation between the lower IT infrastructure and the higher enterprise architecture of services is crucial to understand the exact cause of each critical situation, identify the elements involved, and devise the most suitable mitigation and adaptation plan to reconfigure the system into the desired working state. This can be performed reconciling the information provided by different views into a unified, coherent and complete Run-time Model, with both static and dynamic information about the system, more suitable to respond to internal and environmental events through adaptation strategies.

In addition to these requirements and objectives, the solution should also provide an effective and unified mean to inspect and analyze the system (i.e. the model should be queryable), and facilitate automatic mechanisms of adaptation and reconfiguration (i.e. to enable system-level reflection at execution time). All these requirements have been considered while developing the adaptation engine, by leveraging the architectural knowledge and dynamic information in the model contained in the system specification, as described in detail in the next section.

4 Ontological Engineering

People, organizations, and systems have to communicate and interoperate. However, their different requirements, contexts, and goals are often reflected into a wide range of viewpoints and assumptions, producing different, overlapping and/or mismatched concepts and

[3] http://www.splunk.com/
[4] http://newrelic.com/

structures that however concern the same subject matter. Ontologies are used to overcome this lack of a shared understanding, providing an unifying framework for the different viewpoints that coexist in vast and complex enterprise systems [38].

Ontologies provide many practical benefits, such as serving as the basis for human communication (by reducing conceptual and terminological confusion), improving the interoperability among systems with different modeling methods, paradigms, languages and software tools, and supporting the main system engineering principles (such as reusability, reliability, requirements identification, system specification, etc.).

We initially investigated the possibility of adopting an existing ontology to represent the concepts emerging in complex self-adaptation scenarios, such as our case study. We soon realized that no one single existing solution could fit the diverse contexts we face: domain specific ontologies have the drawback to be too specific, being able to capture only partially and only some of the details of the domain, but omitting others. On the other hands, generic ontologies are too broad, contain details that are not relevant, while other important aspects that should be captured are ignored.

For this reason we propose the use of domain-specific ontologies developed with adaptation in mind. While several approaches are possible, we had good success using the eXtreme Design with Content Ontology Design Patterns (XD) methodology, a collaborative, incremental and iterative method for pattern-based ontology design based on well known ontology engineering principles and best practices described in [39] and [40]. XD identifies an approach, a family of methods, and some associated tools based on the application and exploitation of Ontology Design Patterns (ODPs) [41].

ODPs are modeling solutions enabling the reuse of encoded experiences and good practices to solve recurrent ontology design problems. They can be of different types, including: *logical*, which typically provide solutions for solving problems of expressiveness e.g., expressing n-ary relations in OWL; *architectural*, which describe the overall shape of the ontology (either internal or external) that is convenient with respect to a specific ontology-based task or application e.g. a certain

DL family; *content*, which are small ontologies that address a specific modeling issue, and can be directly reused by importing them in the ontology under development e.g., representing roles that people can play during certain time periods; *presentation*, which provide good practices e.g. naming conventions; etc.

Moreover, XD is partly inspired by eXtreme Programming (XP) [42] and experience factory [43]. Both approaches introduce principles that are relevant to our case study: minimizing the impact of changes at any stage of the development and producing incremental releases based on customer requirements (XP), and improving products quality exploiting past experiences' know-how (experience factory).

5 Systems Engineering

The architecture at the base of our approach splits a system into an *adaptive sub-system* (including the domain logic) and an *adaptation engine* (that controls the system by monitoring its state and effecting changes when needed). This approach implements the adaptation engine as a feedback loop [44] and is opposed to "internal adaptation", which interwines reconfiguration strategies (i.e. sensors, effectors, adaptation processes) and application in a single component. The internal approach has proved to be useful for handling local adaptations (e.g. for exception handling), but has the drawback that often leads to poor scalability and maintainability, as described in [1]. In our external approach, the feedback loop is implemented following the MAPE-K paradigm [6] in four stages (Monitor/Analyze/Plan/Execute) that share the Knowledge of the managed elements, the adaptable software is monitored and analyzed, and if changes are required, adaptation is planned and executed.

The shared knowledge of the system is represented by Reflection Models, which reflect the adaptable software and its environment, mapping system-level observations to a higher level of *abstraction.* Reflection Models accommodate both *static and dynamic* information, and are paired with other models to define, for example, the system expected behavior (Requirements Models), reconfiguration

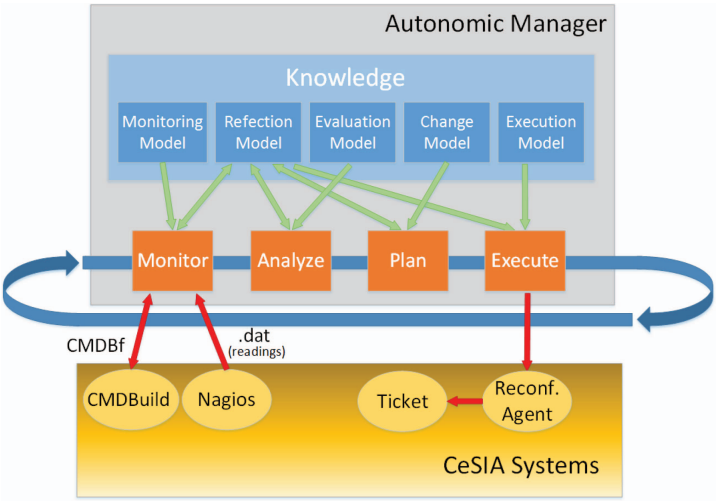

Figure 1 The architecture of the autonomic manager.

policies (Evaluation Models), mappings between reconfigurations and system-level adaptations (Execution Models), etc.

A key point of our solution is using a declarative approach representing this knowledge in an ontological form providing both an architectural view of the system components and their relationships, a description of the system configurations, and a clear definition of the adaptive behaviors associated to each component. The methodology used to develop the semantic models for the case study is sketched in Section 4.

The overall architecture of the autonomic manager is depicted in Figure 1. In the lower part of the picture, a rectangle represents the CeSIA system. Three components are directly involved in the adaptation process: CMDBuild[5] contains structural information about the CeSIA system (i.e. servers, applications, services, and their relationships); Nagios implements the monitoring infrastructure, gathering data on the dynamic behaviors of its components; the Reconfiguration agent collects the output of the autonomic manager, if the needed reconfiguration can be enacted automatically the agent task is to perform it, if the reconfiguration needs human intervention the agent creates tickets

[5]http://www.cmdbuild.org

that are inserted in an issue ticketing system. All managed hosts host worker agents that enact local reconfiguration when requested by the reconfiguration agent (this usually takes the form of starting/stopping services or adding/removing resources to/from running services).

The gray rectangle in the upper part of the picture represents the autonomic manager. The four MAPE-K phases are implemented by concurrent Java components, which coordinate by passing a control token through shared queues. While the current implementation is centralized, the design is meant to be easily replicable in a distributed environment (with obvious advantages in term of robustness and availability).

The shared knowledge is refined to a set of semantic models that mimic the taxonomy presented in [45], with run-time information about the adaptable systems and all the information required to implement any adaptation activities. These models are implemented as OWL ontologies, stored in an external triplestore. This triplestore is the only communication channel used by the autonomic manager components, which perform read and write operations using standard mechanisms (i.e. using SPARQL).

6 Examples from the Case Study

In order to clarify our approach, in this section we describe a running example of an adaptation loop. As discussed in the previous sections, our adaptation engine is modeled around the four MAPE-K phases. The loop frequency varies from context to context, depending on the degree of reactivity required by each system: for the CeSIA adaptation requirements (e.g. react to service delays, replace or reconfigure defective devices, etc.) a frequency of a few minutes is a reasonable value.

The objective of the first MAPE-K phase (*Monitor*) is to provide an updated view of the system and environment state, that will be used in the following steps to identify the adaptation needs and implement the most suitable adjustments. In particular, it is responsible for providing a Reflection Model, which is constituted by *static and dynamic* information about the monitored elements.

The *static information* concerns the structure of the system in terms of logical components and their relations. In our example, they are hardware (e.g. servers, network devices, etc.) and software (e.g. applications, services, etc.) components, and their relationships (e.g. component dependencies, application and business processes, etc.). Among the services provided to UniBo by CeSIA (e.g. mail, backup and storage, computational resources, etc.), in this example we focus on a recurring pattern that is used by CeSIA to model most of the web services exposed at UniBo. This pattern is derived from the Archimate [46] models developed by CeSIA for tracing and navigating between low level services/nodes and high level processes/applications, and has been refined using the methodology described in Section 4. As shown in Figure 2, UniBo exposes to any client some high-level interfaces to its services in the form of web portals (e.g. a main portal for the university and eight sub-sites, one for each school).

The architecture is realized through a flexible cluster implemented on top of a virtualization infrastructure (which imply that all nodes are virtual machines). There are mainly three elements types in the architecture: balancer nodes, cluster nodes, and DBMS nodes. At the top of the picture, two balancer nodes are paired by a fail-over link and use heartbeat for guaranteeing high availability: in case that the primary node becomes unavailable, the secondary replaces it. The traffic load is balanced to the cluster nodes, which have a common structure: Varnish[6] HTTP accelerator is primarily used for caching, then HAProxy[7] balances the load to the Zope Application Server processes, which serve the requests querying the Memcached[8] component. If data and objects are not retrieved in memory, an external data source (i.e. the DBMS node) is interrogated. A copy of this last cluster component is mirrored for disaster recovery, and is managed using Oracle/Microsoft technologies. The information about the elements of this architecture and their relationships is stored in a CeSIA server as CMDBuild assets, and is converted by our Monitor in RDF triples that conform to the CeSIA ontology. To perform this task, a Monitoring Model is used

[6]https://varnish-cache.org/
[7]http://www.haproxy.org/
[8]https://memcached.org/

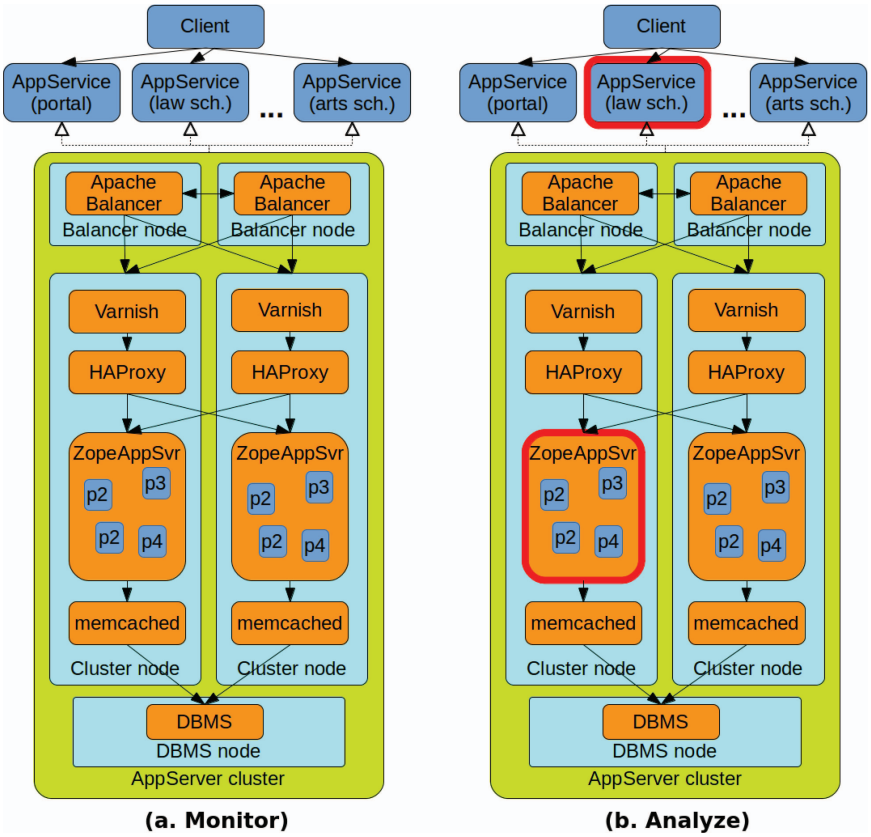

Figure 2 The ontological model for the CeSIA case study during the Monitor (a) and Analyze (b) phases. In this example, the adaptation engine observes an anomalous behaviour (i.e. an high response time) in the portal of the Law School. The cause of the delay is a problem in the Zope Application Server of the left cluster node.

to map these system-level observations to the abstraction level of the Reflection Model.

This static picture reflecting the system structure is enriched with the *dynamic information* (e.g. the response time of an application, the resource distribution of a server, a node temperature, etc.) coming from the sensors of the monitoring system. In particular, the output of Neteye checks are stored on a regular basis in a .dat file, which is read at each stage and mapped using the Monitoring Model to the semantic

level, enriching the Reflection Model with CeSIA systems' dynamic information.

The next phase (*Analyze*) is responsible for recognizing all those critical situations that need a system reconfiguration. The adaptation requirements are identified by applying the Evaluation Model that defines constraints on the Reflection Model. The Evaluation Model contains, for example, a characterization of slow application as such having a response time greater than 10 seconds, of an overloaded server as having more than 95% CPU load, etc. We exploited simple SPARQL queries to periodically interrogate the knowledge base, filter the elements of interest, and add the required statements. Other approaches can be envisioned, as described in [47].

We modeled this information using the *pso:StatusInTime* class included in the Publishing Status Ontology (PSO)[9], which allows to describe the state (e.g. to be slow) that a system component (e.g. the Zope Application Server) has in a given time interval. Each state can be enriched with metadata information: for example, we can add details about the agent that executed the query and the current loop phase. Figure 2(b) shows the updated Run-time Model, where red rectangles highlight the elements interested by the newly added statements. In particular, the Application Service interface of the School of Law and the Zope Application Server are overloaded due to a massive number of incoming requests (i.e. the number of simultaneous incoming requests is very large, and saturates the processes in the server pool).

In the third phase (Plan), the adaptation engine uses both the structural knowledge about the system architecture and the results of the previous analysis on the system behaviors to devise, if needed, a reconfiguration plan. This is accomplished in two consecutive steps. First, the planner uses the dependency relations among the system components to assign a priority to the recognized issues. This task can be accomplished in many ways. For example, different weights or mechanisms such as utility functions [48] can be used to drive the selection process. For our case study, we developed also this part of the engine through SPARQL queries. In particular, the following query

[9]The PSO ontology is available at http://purl.org/spar/pso

implements the principle of selecting as a priority all the defective components that does not depend from other defective ones.

```
SELECT ?start {
  ?start a cesia:Component ;
    pso:holdsStatusInTime/pso:withStatus ?status1 .
  FILTER NOT EXISTS {
    ?end ^cesia:dependsOn+ ?start ;
      pso:holdsStatusInTime/pso:withStatus ?status2 .
  }
}
```

This is an example of how reasoning mechanisms can be leveraged for driving adaptation strategies. Since all the dependency relations in our model have been defined as transitive, we can run a reasoner on the model and automatically derive all the dependency relations. The execution of this query on the semantic model enriched with the inferred relations returns, as we expected, the Zope Application Server (marked with a cross in Figure 3(c)).

The objective of the second step of the Plan phase is to select the most suitable reconfiguration policies for the selected components. These choices are performed using the Change Model that contains the list of the possible reconfigurations for each system element. For example, the CeSIA system defines a list of adaptation strategies in the form of logical rules that are evaluated in sequence on the existing knowledge. The first rule whose premises are satisfied is chosen, and its consequences produce statements that enrich the model with the the strategy to implement. Since the cluster node on the left is not overloaded (i.e. it still has RAM and CPU load available), the selected reconfiguration strategy is to activate more Zope Client processes into the Zope Application Server. In fact, increasing the number of processes in the pool can be a suitable choice to respond to an increased number of requests to the web portal of the School of Law.

The objective of last phase, i.e. *Execution*, is converting the model-level adaptation into the system-level by supervising the needed operations, and synchronizing the Reflection Model in the knowledge base with the new system state. Figure 3(d), for example, shows the

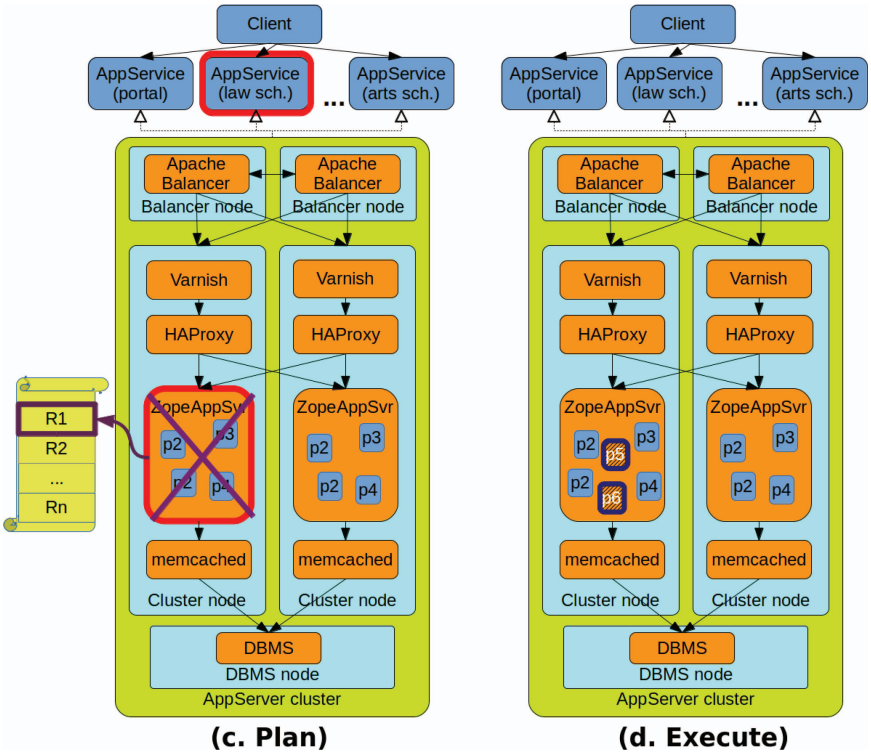

Figure 3 The ontological model for the CeSIA case study during the Plan (c) and Execute (d) phases. In this example, the reconfiguration policy implemented to solve the delays in the portal of the Law School consists in adding client processes in the Zope Application Server.

Zope Application Server pool updated with the added client process (striped with bold borders). Obviously, not all the adaptation strategies can be automatized. For instance, the replacement of broken devices is an activity that needs an human intervention. In all these cases, the Execution phase filters the chosen adaptation strategies that have been devised by the Plan phase reading the Reflection Model, and uses the information read from the Execution Model to instruct the reconfiguration agent or to assign a ticket to the most appropriate human agent. In both cases, all the information about the state of the managed elements are updated into the knowledge base, ready to be processed by the next cycle of the adaptation loop. Further adaptation actions can

take place only after a transient period meant to accommodate for the set up of the new components. It is the so called *warm-up time*, during which machines are booted, programs are started, data structures are populated and so on.

The monitoring and management of web portals problems (such as delays) is one of the most common activity at CeSIA. The adaptation strategy described in the previous example is the automatization of the most common reconfiguration strategies used by CeSIA. During the design of our approach, we searched for a trade-off between system expressiveness (in terms of both system architectures and adaptation strategies) and complexity. In particular, we tried to minimize the effort required to design, add and test new reconfiguration strategies.

Another very common reconfiguration scenario that we successfully managed with our approach is shown in Figure 4(a). In this case, the delays perceived by the clients of the School of Law portal are caused by a problem in the cluster node, that has an average CPU load and memory usage greater than 90%.

In this case, a reasonable adaptation strategy is deploying another cluster node, which translates in the activation of a new virtual machine (as long as the private cloud infrastructure has resources to host additional virtual machines), as shown in Figure 4(b). This approach has the advantage of requiring no node downtime while improving the performance with the addition of caching and balancing components to the cluster architecture. Since the architecture of the Application Server Cluster is virtualized, also other "light-weight" approaches can be devised, such as guaranteeing more resources (e.g. CPUs and memory) to a running virtual machine serving as cluster node.

The chosen adaptation strategy can be achieved by adding a new reconfiguration rule in the Change Model that will be evaluated during the Plan phase, and extending the Execution Model with instructions about the operations to enact the planned system changes. The deployment/undeployment of an additional cluster node in our virtual infrastructure requires a minimal administrative effort and is performed by the reconfiguration agent using the hypervisor APIs.

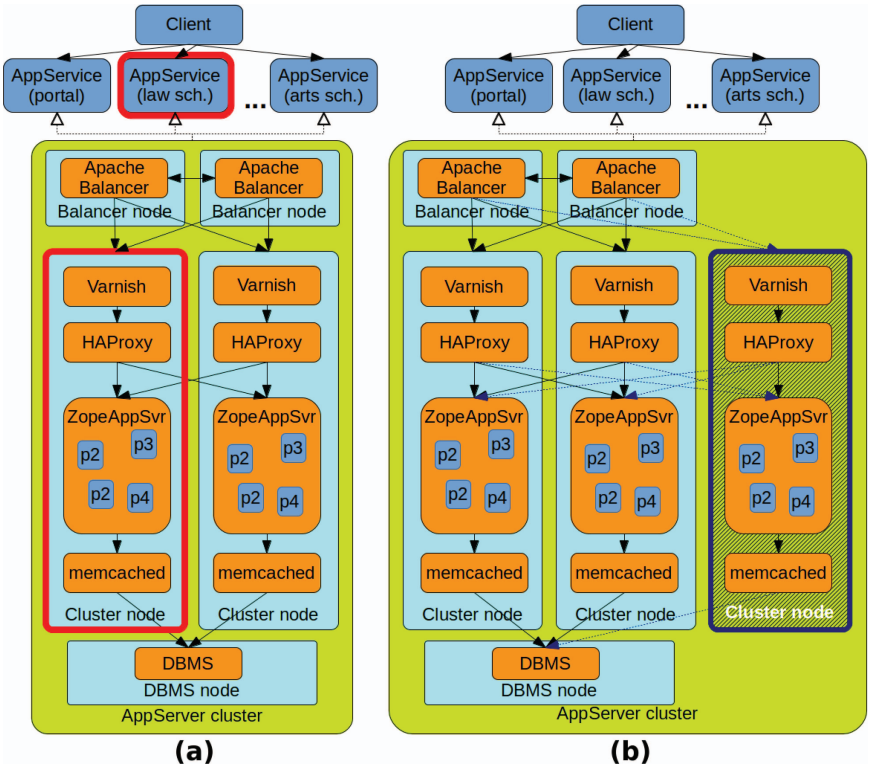

Figure 4 In this example, the delays in the Law School portal are caused by a problem in a cluster node (a). The chosen adaptation policy is adding another node to the cluster (b).

7 Experiments

The system described in Section 5 has been built instantiating the policies explained in Section 6. We created a copy of the UniBo portal (with a subset of its contents) using the very same virtualization infrastructure and the same software components. All nodes were set up to host a reconfiguration worker and the autonomic manager (including the triplestore) hosted in a separate virtual machine. The tests have been set up aiming at verifying the ability of the system to trigger adaptation activities as described in the examples of Section 6 when

the performances of the portal are not aligned with a given Service Level Agreement (SLA). The SLA mainly consists in upper bounds on the response times for various resources sets and an allowed percentage of violations in a given time window.

The various components of the autonomic manager have been configured to interact with the knowledge base via SPARQL statements used to create and query RDF triples, implementing the needed policies. The following SPARQL query, for example, filters all the application servers that are slow, that have both a CPU load and memory usage less than 85%, and that are running a number of processes which is less than the number of available cores.

```
SELECT ?appserver ?label ?rt
WHERE {
    ?appserver a cesia:ApplicationServer ;
      rdfs:label ?label ;
      pso:holdsStatusInTime/pso:withStatus
            cesiadata:slowComponent ;
      cesia:hostedBy+ ?node .
    ?node a cesia:Node ;
      cesia:numCores ?numCores .
      cesia:hasObservation ?obs .
    ?obs cesia:hasObservedResponseTime ?rt ;
      cesia:hasObservedCpuLoad ?cl ;
      cesia:hasObservedMemoryUsage ?mu ;
      cesia:hasObservedProcesses ?p .
      cesia:hasObservedResponseTime ?rt .
    FILTER(?cl <= 85^^xsd:decimeal)
    FILTER(?mu <= 85^^xsd:decimeal)
    FILTER(?p < ?numCores)
} ORDER BY DESC(?rt)
```

By running this query, we get the list of all the application servers that can be reconfigured by adding a new process, in descending order response time.

```
appserver                     |label                          |rt
------------------------------+-------------------------------+----
cesiadata: zopeappserver -2   |Plone v5.0 - School of Law #2  |945
cesiadata: zopeappserver -1   |Plone v5.0 - School of Law #1  |711
cesiadata: zopeappserver -25  |Plone v4.4 - School of Arts #3 |689
```

Other reconfiguration strategies have been devised for our case study. For instance, when the previous policy cannot be applied (e.g. because the CPU of the node that hosts the slow component is overloaded), the autonomic manager checks whether it is possible to add a new node to the cluster. The following excerpt implements this policy by filtering all the application clusters that contains slow application servers and that have not reached their maximum capacity.

```
SELECT DISTINCT ?cluster ?label ?maxNodes ?currNodes
WHERE {
    ?appserver a cesia:ApplicationServer ;
        redfs:label ?label ;
        pso:holdsStatusInTime/pso:withStatus
            cesiadata:slowComponent ;
        cesia:hostedBy+/odp:isPartOf ?cluster .
    ?cluster a cesia:Cluster ;
        cesia:nodeLimit ?maxNodes .
    {
        SELECT ?cluster (count(?parts) as ?currNodes)
        WHERE {
            ?cluster a cesia:Cluster ;
                cesia:hasPart ?parts
        } GROUP BY ?cluster
    }
    FILTER(?currNodes <= ?maxNodes)
}
```

This second SPARQL query produces the list of the clusters that can be reconfigured by adding a new cluster node, as shown in the following excerpt.

```
cluster              | label                      | maxNodes  | currNodes
---------------------+----------------------------+-----------+---------
cesiadata: cluster-7 | School of Eng.-Web Portal | 4         | 2
```

All the reconfiguration policies are contained in the Change Models in the form of a list of SPARQL queries similar to the previously described ones. In order to select the strategy to apply, the autonomic manager runs the queries in sequence on the Reflection Models, and selects the first that returns a non-empty set. If more than one element is returned, only the first is chosen as the next to be reconfigured. The information about the adaptation policy that needs to be carried out are attached to the selected node, and added to the Reflection Models in the KB, ready to be performed during the next Execution phase. If no reconfiguration can be applied, the autonomic manager performs the most general reconfiguration policy, that consists in tracing this situation in the Reflection Models, and notify the systems administrators about the issue.

The tests have been performed using an artificial load generator able to mimic the access patterns of the users as extracted from the running portal logs. We then used this generator to inject requests as coming from a raising number of users and checked how the system performed self-adaptation activities in order to meet the SLA. The architecture of such system is described in details in Section 6.

The initial cluster is composed by two nodes with four cores and two gigabytes of RAM. Two Zope Client processes are loaded into the application servers. Moreover, we decided to turn varnish off, since we wanted to exclude the caching system from our experiment and mainly focus on the other components of the architecture.

Before starting the experiment, we also set some configuration parameters within the autonomic manager. We set the maximum number of processes within each application server to four in order to not exceed the total number of processors in each node, a best practice suggested by the application server developers. We also set the maximum number of concurrent cluster nodes to six in order to not exceed the limit imposed by the resources (e.g. CPUs and memory) available for the virtualization infrastructure. The frequency of the adaptation loop is set to 10 s. Finally, in order to meet the SLA, we set

the response time threshold to 600 ms. When the cluster node exceeds this limit, the autonomic manager intervenes by reconfiguring the cluster.

We prepared a list of twenty HTML pages within our test site. The address of each request is randomly selected within this page pool. We started keeping 10 concurrent open HTTP connections, and raised linearly until we reached a maximum of 80 connections in 20 minutes. This load has then been kept for five additional minutes. All the requests are performed by a single machine within the application cluster subnetwork, connected to the physical machine that hosts the cluster through a switch with gigabit connections (we also performed some distributed tests to make sure that the single load-injecting machine was not a bottleneck and it turned out it was not).

In order to evaluate our autonomic manager, we performed the experiments twice, first with the autonomic manager turned off and then on. In the first run of the experiment, the response time of the web portal increases linearly as the number of concurrent connections grows, settling around a value of 1.8 seconds (see Figure 5). The number of page hits/second is stable at around 4500 throughout the duration of the experiment, since the number of concurrent requests (10 or more for the whole duration of the experiment) exceeds the available resources (4 serving processes, 2 on each cluster node).

The results of the experiment with the autonomic manager turned on is depicted in Figure 6. In this case, the response time grows linearly until it reaches the threshold of 600 ms after around 5 minutes from the beginning of the experiment. At this point the autonomic manager applies the first adaptation policy by running new process into the two application servers.

After four reconfigurations (the spikes in the response time line, approximately one every 50 seconds), the two application servers reached the maximum number of runnable processes. For this reason, around minute 8:30, a new node with two processes is added to the cluster, producing another spike and then a fall in the response time line. Two more processes are added to the newly added node, and then at around 14:30 a fourth node is added to the cluster. No other

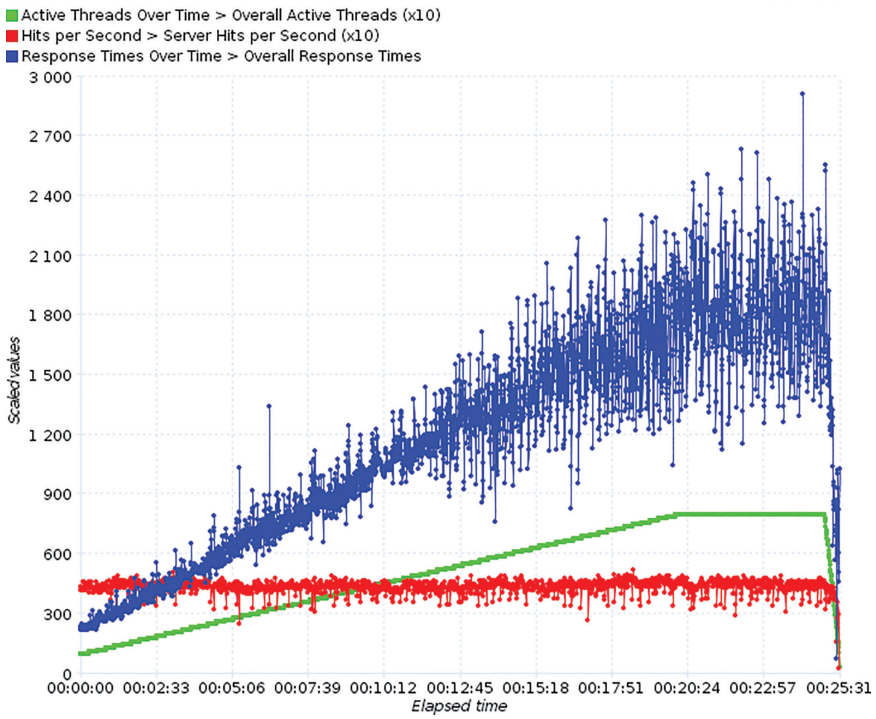

Figure 5 Number of active threads over time, response time and hits per seconds of our experiment with the autonomic manager turned off.

reconfigurations are performed until the end of the experiment, since the response time sensed by the autonomic manager is always under or around the threshold of 600 ms. Another interesting aspect worth noting is the number of hits per second, which increases at each reconfiguration (as we expected), rising more evidently when new nodes are added to the cluster (the two spikes at 5:10 and 14:30).

As previously stated these tests are aimed at verifying the autonomic adaptation capability of the system in presence of a rather simple load pattern; we were not, at this stage, interested in sophisticated elasticity algorithms (such as for instance those described in [49]) designed to face more complex loads, such as flash crowds.

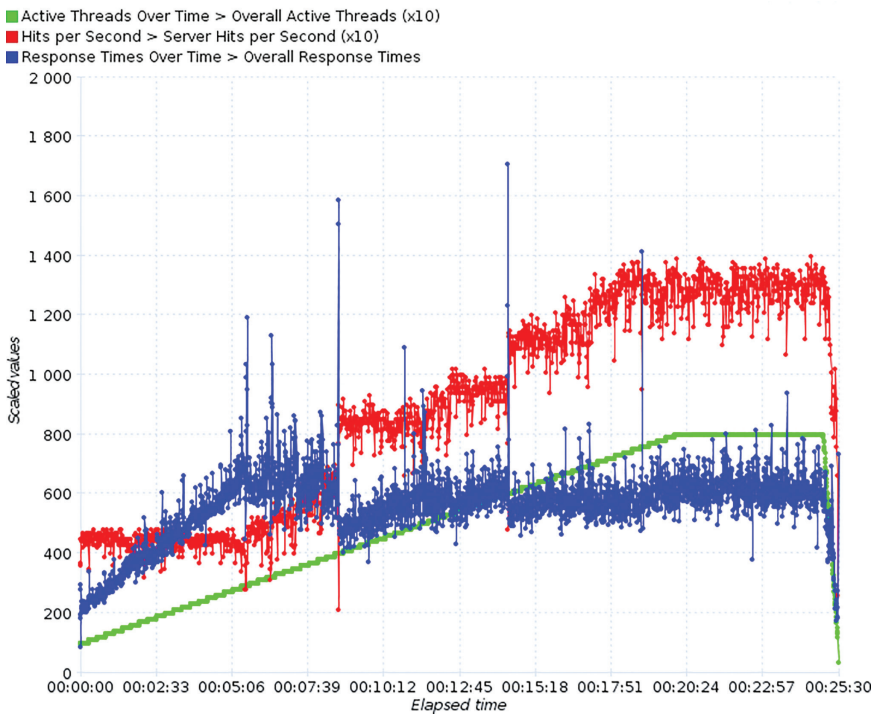

Figure 6 Number of active threads over time, response time and hits per seconds of our experiment with the autonomic manager turned on.

However, we performed also an experiment with a more challenging load pattern that corresponds to a typical event in the operations of our real-world case study: sudden and sustained increase access rate due to specific events. A typical example is the access rate of the students enrollment service right after the opening date (this usually happens in late August). In such a case we witness a sudden raise in the request rate; with our implementation this would trigger the various self-adaptation activities bringing the system into a state in which it is able to sustain the requests while respecting the SLA. However, the requests received while the reconfiguration activities are in progress would experience delays and, in same cases, failures. The solution currently adopted to overcome this problem is to rely upon over-provisioning: the system is configured with more cluster node replicas and more server processes

than needed by the usual number of users so that when the load suddenly increases the system can serve all the requests with no excessive delays or errors. This is a quite expensive policy and is adopted by hand-made reconfiguration performed by the administrators when they know that specific dates are approaching.

Replicating this situation with a self-adaptive mechanism has been quite easy with our approach: it is just matter of integrating a calendar of critical dates in the knowledge base and add new time-triggered over-provisioning policies (expressed with SPARQL statements); both adjustments can be performed on-the-fly with no (or very limited, as in the case of the addition of a new data source) system downtime. Figures 7 and 8 shows the behaviors of the cluster with and without over-provisioning, respectively.

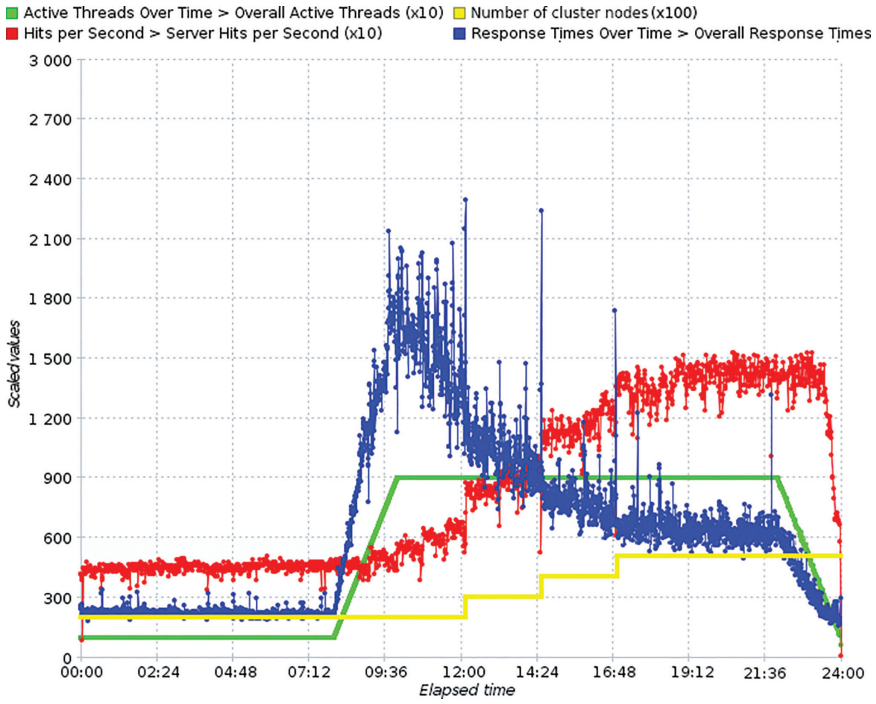

Figure 7 Without over-provisioning, the application cluster does not respect the SLA (the response time raises around 2 s. when the maximum load is reached, and needs time to add nodes and reconfigure the system).

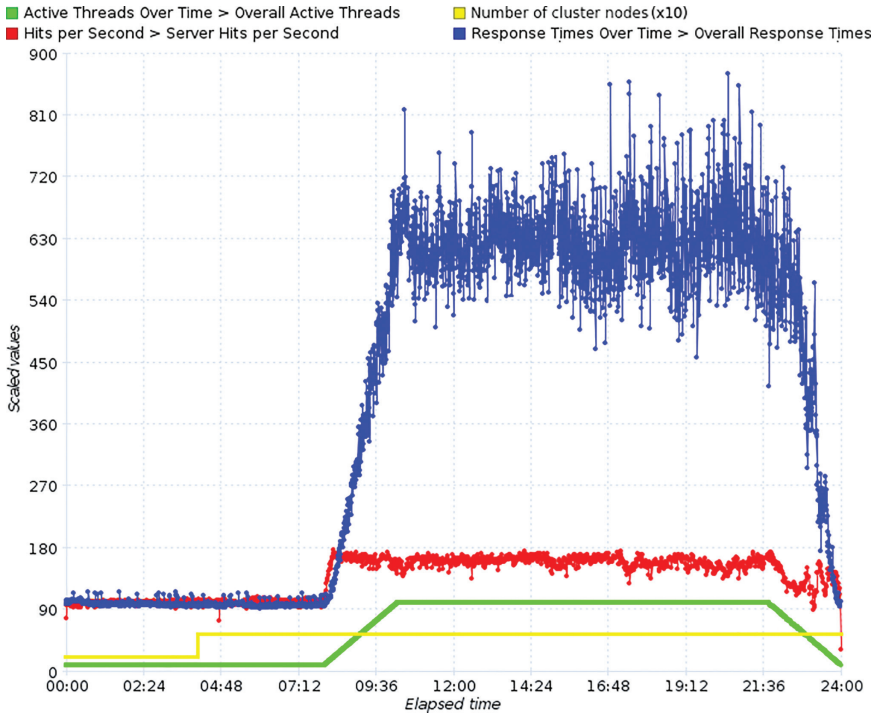

Figure 8 With over-provisioning (three additional cluster nodes have been added), the application cluster meets the SLA and stabilizes around a response time of 600 ms.

8 Discussion and State of the Art

In this work we faced two main challenges to model-based adaptation: finding a unified framework to manage the heterogeneity of real-world systems, and designing models queryable at run-time suitable to support reasoning about the states of the system they model. The key point of our solution is using Semantic Web technologies and some related tools to represent and manage adaptable systems models both at design and run time. The design and deployment of a working self-adaptable system in a real-world use case has provided us with a chance to investigate the potential of our approach, showing that it can be successfully implemented in a real operational context.

While it can easily be argued that what can be done using this approach can be accomplished as well in other ways, we believe that

our proposal lowers significantly the complexity of both design and maintenance activities. This is important to convince industry and institutions that methods based on Model Driven Engineering constitute a viable solution [50].

Szvetits et al. [51] comprehensively survey the approaches that have been proposed to model adaptive systems and that focus on the common idea of establishing semantic relationships between executed

Table 1 Comparison of our work with other main approaches to adaptation based on runtime models at the state of the art

Paper	Kind of Run-time Models	Tech/Syntax/ Approach	Main Objective
Whittle et al. [52]	Requirements Models	Fuzzy Branching Temporal Logic	Requirements change
Inverardi and Mori [54]	Feature Models, Context Models	Probabilistic Automaton	Contextual Change
Calinescu et al. [56]	Evaluation Models	Markov models (i.e. stochastic processes)	Reliability (NFR)
Bromberg et al. [57]	Behaviour Models	K-Colored and Merged Automata	Interoperability (NFR)
Schneider and Trapp [58]	Safety Models	Conditional Safety Certificates [64] (Binary Decision Diagrams [65])	Efficiency (NFR)
Röttger and Zschaler [60]	Context Models	CQML+ [66]	Usability (NFR)
Georgas et al. [62]	Architecture Models	Directed Cyclic Graph	Monitoring
Ales and Matjaz [63]	Business process model	BPMN (extended)	Policy Checking and Enforcement
This paper	Semantic Models	OWL	Adaptation

applications and run-time models. In Table 1 we summarize the main approaches, and compare them with our solution based on semantic run-time models. In particular, we describe the main adaptation problems addressed and the kind of run-time model(s) at the base of each work, and provide details about the technologies used.

One of the main problem faced by adaptive systems is to meet requirements changes and evolution in operational environments (i.e. application contexts). Main challenges in the development of adaptative systems lie in finding out and analyzing alternative configurations, comparing their benefits with the current system state. Models at run-time help to model variable parts of the system during the system execution, and help the development of a formal representation of requirements that can be processed at run-time. The requirements language RELAX [52], for instance, is a declarative language for self-adaptive systems which supports the explicit expression of environmental uncertainty in requirements. RELAX is based on fuzzy branching temporal logic and provides modal, temporal and ordinal operators to express uncertainty. Example operators are SHALL, to define functionality the system must always provide (invariants), and MAY/OR to define alternatives. In [53, 37] we extended the work presented in this paper by introducing requirements models to our semantic run-time models, adding requirement-awareness capabilities to our approach. Experimental results show that modification of the requirements models at run-time result in different re-configuration decisions of our adaptive engine.

Inverardi and Mori [54] focus on the problem of continuously changing contexts and requirements at run-time, and propose a software development process to support consistent evolution with the help of Context Models, Features Models, and a control loop. Their framework can be applied at different levels of abstraction spanning from code to software architecture, and is based on probabilistic automaton theory to drive adaptation.

The management of goals expressing Non-Functional Requirements (NFRs) – i.e. those requirements for which satisfaction cannot be established in a clear-cut sense, such as reliability, availability, security, etc. – is another of the main problems faced by current researches on

adaptive systems [55]. Calinescu et al. [56], for instance, propose a framework named QoSMOS which utilizes Markov models within a feedback loop to quantify the performance and reliability of service-based systems. The monitoring information are evaluated with the help of specified QoS requirements, and reconfiguration plans are defined in terms of changes to resource allocation policy and/or to the overall application workflows.

Interoperability of distributed adaptive systems is the main challenge faced by Starlink [57]. In environments where services and systems are composed dynamically, e.g. pervasive computing and Systems-of-Systems, the protocols used by two systems are usually not known until run-time. The main component of the proposed framework is an engine that executes coloured automata representing the interoperability behaviors between protocols, and translates message content from one protocol to another. In [58], the authors introduce the notion of Safety Models to support the management of dynamic behaviors of system at run-time so that efficiency requirements are met in evolving system configurations. An approach for the development of reactive systems based on semantic models and inspired to agile methodologies which considers both functional and non-functional requirements is presented in [59].

To improve tool support for NFRs management, Röttger and Zschaler [60] propose the use of Context Models to make the specification of non-functional measurements (i.e. nonfunctional dimension that can be constrained to describe a non-functional property) independent of their use in concrete system specifications. Transformations between context models and measurements allows the application designer to focus on the business logic when developing an application.

We initially analyzed NFRs in adaptive systems based on semantic run-time models in [47, 61], and them managed them introducing Softgoals Models in [53].

Georgas et al. [62] face the problem of monitoring adaptive systems, and propose an approach based on a graph representation of the components that capture historical configurations and corresponding system behaviours. The adaptive process is augmented with metadata such as frequency/duration of entered configurations and context information

(Context Models). A run-time Architecture Model is constituted by the historical graph consisting of system configurations (nodes) and transitions (edges), where transitions store architectural differences between configurations in a bidirectional way, enabling rollback and rollforward operations. Another interesting work [63] is based on business process models describing sequences of business activities that are used to check workflow conformance and guide adaptation through extensions to BPMN (Business Process Model and Notation) elements.

9 Conclusions and Threats to Validity

Changes in environment-related aspects and user needs are the challenges that (self-)adaptive systems have to deal with. These systems drive their behavior on the basis of a knowledge that is composed of models representing facts pertaining the environment, the systems itself, the user needs. In this paper we presented an approach to manage this knowledge under a unified semantic architecture. We put this architecture at work by creating a prototype system based on it. Our tests show that specific real-world critical systems can be fruitfully implemented using our approach. Not only the resulting system is in fact able to perform specific adaptation activities in an autonomous way but we have been able to achieve this goal with a simple, clean, easy to understand and maintain infrastructure that can easily (and on-the-fly) accommodate new sources of information and policies' changes.

We evaluated our approach in a specific application field. In order to evaluate our approach more broadly we will focus on the adaptation engine. In particular, we plan to compare different reconfiguration strategies, and study how and to what extent the system gets affected by these automated interventions.

We now discuss some potential threats to the validity of our approach in various contexts. Here is a list of the issues we perceive as the most relevant.

Performance: while triplestore datasets have reached a level of robustness and performance close to that of relational databases, current semantic inference engines are not as well performing as their business

rules engines counterparts. Several research projects are currently working to overcome these limits. For example, stream reasoning [67, 68] techniques have already shown that the use of semantic technologies for the management of large streams of data is quite effective.

Design vs. run-time tracing: tracing between models is a relevant issue, mostly with model-driven engineering techniques. While we still have not implemented a specific solution, we observe that the ability to easily tag any element of our semantic models should reasonably simplify the task.

Requirement model representation: different policies [48] (situation-action vs goal-oriented vs utility function-based) need different refinement and representation of the requirements. In some cases also uncertainty [52] has to be taken into account. This could lead to different modeling strategies and impact some adaptation activities. With our running use case we investigated a goal-oriented approach. While it is perfectly reasonable to assume that the use of semantic models for different kinds of requirements is indeed possible, the level of complexity needed to link requirements and system artifacts could be not trivial.

Acknowledgments

This paper was supported by the MIUR-PRIN GAUSS Project, and by the Consorzio Interuniversitario per l'Informatica (CINI) under projects EMC2 and MANTIS.

References

[1] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, p. 14, 2009.

[2] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback control of computing systems.* John Wiley and Sons, 2004.

[3] K. J. Aström and R. M. Murray, *Feedback systems: an introduction for scientists and engineers.* Princeton university press, 2010.

[4] G. Blair, N. Bencomo, and R. B. France, "Models@run.time," *Computer*, vol. 42, no. 10, pp. 22–27, 2009.
[5] P. Oreizy, M. Gorlick et al., "An architecture-based approach to self-adaptive software," *IEEE Intelligent systems*, no. 3, pp. 54–62, 1999.
[6] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.
[7] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, 2004.
[8] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Future of Software Engineering*. IEEE Computer Society, 2007, pp. 37–54.
[9] M. Derakhshanmanesh, J. Ebert, M. Grieger, and G. Engels, "Model-integrating development of software systems: a flexible component-based approach," *Software and Systems Modeling*, 2018.
[10] P. Agarwal, "Ontological considerations in giscience," *International Journal of Geographical Information Science*, vol. 19, no. 5, pp. 501–536, 2005.
[11] W. O. W. Group, "Owl 2 web ontology language," 2012. [Online]. Available: https: //www.w3.org/TR/owl2-overview/
[12] E. Prud'Hommeaux, A. Seaborne *et al.*, "Sparql query language for rdf," 2008. [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/
[13] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," *Web Semantics: science, services and agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, 2007.
[14] D. Tsarkov and I. Horrocks, "Fact++ description logic reasoner: System description," in *Automated reasoning*. Springer, 2006, pp. 292–297.
[15] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, "HermiT: an OWL 2 reasoner," *Journal of Automated Reasoning*, vol. 53, no. 3, pp. 245–269, 2014.

[16] I. Horrocks, Patel-Schneider *et al.*, "SWRL: A semantic web rule language combining OWL and RuleML," 2004.

[17] J. Rao and X. Su, "A survey of automated web service composition methods," in *Semantic Web Services and Web Process Composition.* Springer, 2005, pp. 43–54.

[18] D. Martin, M. Burstein, D. Mcdermott, S. Mcilraith, M. Paolucci, K. Sycara, D. L. Mcguinness, E. Sirin, and N. Srinivasan, "Bringing semantics to web services with OWL-S," *World Wide Web*, vol. 10, no. 3, pp. 243–277, 2007.

[19] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. Martin, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and others, "DAML-S: Web service description for the semantic web," in *The Semantic Web—ISWC 2002*. Springer, 2002, pp. 348–363.

[20] S. Narayanan and S. McIlraith, "Simulation, verification and automated composition of web services," in *Proc. 11th Int. Conf. on the WWW*. ACM, 2002, pp. 77–88.

[21] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan, "Automated discovery, interaction and composition of semantic web services," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 1, pp. 27–46, 2003.

[22] E. Sirin, J. Hendler, and B. Parsia, "Semi-automatic Composition of Web Services using Semantic Descriptions," *In Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS. 2003.* Citeseer, 2002.

[23] G. Gharbi, M. B. Alaya, C. Diop, and E. Exposito, "Aoda: an autonomic and ontology-driven architecture for service-oriented and event-driven systems," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 13).* IEEE, 2012, pp. 72–77.

[24] S. Ramanathan, A. Kamoun, and C. Chassot, "Ontology-based collaborative framework for disaster recovery scenarios," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 13).* IEEE, 2012, pp. 104–106.

[25] S. Niemczyk and K. Geihs, "Adaptive run-time models for groups of autonomous robots," in *Proc. 10th Int. Symp. on Software*

Engineering for Adaptive and Self-Managing Systems, 2015, pp. 127–133.

[26] M. Gelfond and V. Lifschitz, "Classical negation in logic programs and disjunctive databases," *New generation computing*, vol. 9, no. 3–4, pp. 365–385, 1991.

[27] G. Stevenson, J. Ye, S. Dobson, D. Pianini, S. Montagna, and M. Viroli, "Combining Self-organisation, Context-awareness and Semantic Reasoning: The Case of Resource Discovery in Opportunistic Networks," in *Proc. 28th ACM Symposium on Applied Computing*, ser. SAC. New York, USA: ACM, 2013, pp. 1369–1376.

[28] H. Chen, F. Perich, T. Finin, and A. Joshi, "Soupa: Standard ontology for ubiquitous and pervasive applications," in *First Int. Conf. on Mobile and Ubiquitous Systems: Networking and Services*. IEEE, 2004, pp. 258–267.

[29] S. Kaebisch and T. Kamiya, "Web of Things (WoT) thing description," *First Public Working Draft, W3C*, 2017.

[30] A. Haller, K. Janowicz, S. Cox, D. Le Phuoc, K. Taylor, and M. Lefrançois, "Semantic sensor network ontology," *W3C Recommendation*, W3C, 2017.

[31] K. Janowicz, A. Haller, S. J. Cox, D. Le Phuoc, and M. Lefrançois, "Sosa: A lightweight ontology for sensors, observations, samples, and actuators," *Journal of Web Semantics*, 2018.

[32] S. Sagar, M. Lefrançois, I. Rebai, M. Khemaja, S. Garlatti, J. Feki, and L. Médini, "Modeling Smart Sensors on top of SOSA/SSN and WoT TD with the Semantic Smart Sensor Network (S3N) modular Ontology," in *Proc. 9th International Semantic Sensor Networks Workshop*, Monterey, USA, 2018.

[33] R. Dautov, I. Paraskakis, and D. Kourtesis, "An ontology-driven approach to self-management in cloud application platforms," in *Proc. 7th South East European Doctoral Student Conference*, 2012, pp. 539–550.

[34] M. B. Alaya and T. Monteil, "Frameself: an ontology-based framework for the self-management of machine-to-machine systems," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 6, pp. 1412–1426, 2015.

[35] R. Dautov, D. Kourtesis, I. Paraskakis, and M. Stannett, "Addressing self-management in cloud platforms: a semantic sensor web approach," in *Proc. Int. workshop on Hot topics in cloud services*. ACM, 2013, pp. 11–18.

[36] R. Dautov, I. Paraskakis, and M. Stannett, "Utilising stream reasoning techniques to underpin an autonomous framework for cloud application platforms," *Journal of Cloud Computing*, vol. 3, no. 1, p. 13, 2014.

[37] D. Rossi, F. Poggi, and P. Ciancarini, "Dynamic high-level in self-adaptive systems," in *Proc. 6th Int. Conf. on Reliability, Infocom Technologies and Optimization (ICRITO)*. IEEE, 2017, pp. 49–60.

[38] M. Uschold and M. Gruninger, "Ontologies: Principles, methods and applications," *The knowledge engineering review*, vol. 11, no. 02, pp. 93–136, 1996.

[39] A. Gómez-Pérez, M. Fernández-López, and O. Corcho, *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer Science and Business Media, 2006.

[40] M. C. Suárez-Figueroa, A. Gómez-Pérez, E. Motta, and A. Gangemi, *Ontology engineering in a networked world*. Springer Science and Business Media, 2012.

[41] A. Gangemi and V. Presutti, "Ontology design patterns," in *Handbook on ontologies*. Springer, 2009, pp. 221–243.

[42] J. Shore et al., *The art of Agile development*. O'Reilly Media, 2007.

[43] V. R. Basili, G. Caldiera, and H. D. Rombach, "Experience factory," *Encyclopedia of software engineering*, pp. 469–476, 1994.

[44] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg, "Models@ run. time to support dynamic adaptation," *Computer*, vol. 42, no. 10, pp. 44–51, 2009.

[45] T. Vogel and H. Giese, "Model-driven engineering of self-adaptive software with EUREMA," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 8, no. 4, p. 18, 2014.

[46] M. M. Lankhorst, H. A. Proper, and H. Jonkers, "The architecture of the archimate language," in *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2009, pp. 367–380.

[47] F. Poggi, D. Rossi, P. Ciancarini, and L. Bompani, "Semantic runtime models for self-adaptative systems: a case study," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 16).* IEEE, 2016.

[48] J. O. Kephart and W. E. Walsh, "An artificial intelligence perspective on autonomic computing policies," in *Proceedings. 5th IEEE Int. Workshop on Policies for Distributed Systems and Networks*, 2004, pp. 3–12.

[49] G. Lodi, F. Panzieri, D. Rossi, and E. Turrini, "Sla-driven clustering of qos-aware application servers," *Software Engineering, IEEE Transactions on*, vol. 33, no. 3, pp. 186–197, 2007.

[50] E. Luna et al., "Challenges for the adoption of Model-Driven Web Engineering approached in industry," *Journal of Web Engineering*, vol. 17, no. 3-4, pp. 183–205, 2018.

[51] M. Szvetits and U. Zdun, "Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime," *Software and Systems Modeling*, vol. 15, no. 1, pp. 31–69, 2016.

[52] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel, "Relax: Incorporating uncertainty into the specification of self-adaptive systems," in Proc. *17th IEEE Int. Requirements Engineering Conference.* IEEE, 2009, pp. 79–88.

[53] D. Rossi, F. Poggi, and P. Ciancarini, "Dynamic high-level requirements in self-adaptive systems," in *Proc. 33rd ACM Symposium on Applied Computing.* ACM, 2018, pp. 128–137.

[54] P. Inverardi and M. Mori, "A software lifecycle process to support consistent evolutions," in *Software Engineering for Self-Adaptive Systems II.* Springer, 2013, pp. 239–264.

[55] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach," *IEEE Transactions on Software Engineering*, vol. 18, no. 6, pp. 483–497, 1992.

[56] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimization in service-based systems," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 387–409, 2011.

[57] Y.-D. Bromberg, P. Grace, and L. Réveillère, "Starlink: runtime interoperability between heterogeneous middleware protocols," in *2011 31st International Conference on Distributed Computing Systems*. IEEE, 2011, pp. 446–455.

[58] D. Schneider and M. Trapp, "Conditional safety certification of open adaptive systems," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 8, no. 2, p. 8, 2013.

[59] P. Ciancarini, A. Messina, F. Poggi, and D. Russo, "Agile knowledge engineering for mission critical software requirements," in *Synergies Between Knowledge Engineering and Software Engineering*. Springer, 2018, pp. 151–171.

[60] S. Röttger and S. Zschaler, "Tool support for refinement of non-functional specifications," *Software and Systems Modeling*, vol. 6, no. 2, pp. 185–204, 2007.

[61] F. Poggi, D. Rossi, P. Ciancarini, and L. Bompani, "An application of semantic technologies to self adaptations," in *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*. IEEE, 2016, pp. 1–6.

[62] J. C. Georgas, A. van der Hoek, and R. N. Taylor, "Using architectural models to manage and visualize runtime adaptation," *Computer*, vol. 42, no. 10, 2009.

[63] A. Frece and M. B. Juric, "Modeling functional requirements for configurable content-and context-aware dynamic service selection in business process models," *Journal of Visual Languages and Computing*, vol. 23, no. 4, pp. 223–247, 2012.

[64] D. Schneider and M. Trapp, "Conditional safety certificates in open systems," in *Proc. 1st Workshop on critical automotive applications: robustness and safety.* ACM, 2010, pp. 57–60.

[65] S. B. Akers, "Binary decision diagrams," *IEEE Transactions on Computers*, no. 6, pp. 509–516, 1978.

[66] S. Röttger and S. Zschaler, "CQML+: Enhancements to CQML," in *Proc. 1st Int. Workshop on Quality of Service in Component-Based Software Engineering*, 2003, pp. 43–56.

[67] A. Margara, J. Urbani, F. van Harmelen, and H. Bal, "Streaming the web: Reasoning over dynamic data," *Web Semantics: Science, Services and Agents on the WWW*, vol. 25, pp. 24–44, 2014.

[68] F. Corcoglioniti, M. Rospocher, M. Mostarda, and M. Amadori, "Processing Billions of RDF Triples on a Single Machine Using Streaming and Sorting," in *Procs 30th ACM Symposium on Applied Computing, ser.* SAC '15. New York, NY, USA: ACM, 2015, pp. 368–375.

Biographies

Francesco Poggi is a Research fellow at at the Department of Computer Science and Engineering (DISI) of the University of Bologna. He holds a Ph.D. in Computer Science from the University of Bologna, since 2015. He won the best paper award at DOCENG 2015 for his paper titled "Exploring scholarly papers through citations". He has been principal investigator for the MIUR-ANVUR of the funded research project "Uniform Representation of Curricular Attributes". His research interests include: adaptive systems and reflective enterprise software architectures; information visualization; Semantic Web technologies; markup languages for complex documents; science of science.

Davide Rossi is an Assistant Professor at at the Department of Computer Science and Engineering (DISI) of the University of Bologna. His activity mainly focuses on applied aspects related to software engineering (modeling, distributed software architectures, middleware, web engineering) with specific interest toward the concepts of service, composition, interaction and process. He participated in several national/international research projects and is the author of more than fifty published contributions in the form of journal articles, international conference/workshop proceedings papers and book chapters.

Paolo Ciancarini is Professor of Computer Science at the University of Bologna since 1992. He got a Phd in Informatics at the University of Pisa in 1988. In Bologna he lectures on Software Engineering and Software Architecture, and is member of the Faculty of the PhD School in Computer Science.