An Ontology-driven Approach to Self-management in Cloud Application Platforms

Rustem Dautov, Iraklis Paraskakis, Dimitrios Kourtesis

South-East European Research Centre (SEERC),
City College – International Faculty of the University of Sheffield,
24 Proxenou Koromila Street, 54622 Thessaloniki, Greece
{rdautov, iparaskakis, dkourtesis}@seerc.org

Abstract. Cloud application platforms, with their support for elastic resources and software development lifecycle, are attracting more and more users wishing to host web-based applications that have variable demand, yet consistent performance requirements. Effective platform and application management is mandatory in order for applications running in the cloud to be cost-effective. We see self-management as one of the way of achieving this goal. In this paper we present our vision of an ontology-driven approach to self-management in cloud application platforms. In the context of the MAPE-K reference model for adaptation cycles, we describe main advantages of using ontologies and rules, in order to represent self-reflective knowledge and define adaptation policies in comparison to more traditional approaches.

Keywords: Ontology, self-management, cloud application platform, adaptation, self-reflection.

1 Introduction

Cloud application platforms are significantly reducing the effort required to create a Web application and deploy it to a cloud, and are completely eliminating the effort to maintain a hosting environment [1]. Managing the environment and ensuring that applications operate at a satisfactory level becomes a concern of the platform [1]. In most cases, to ensure that appropriate service levels are observed, cloud platforms are continuously monitoring the usage of deployed applications and are capable of reacting to important changes in the environment by carrying out some forms of automatic adaptation [1]. Those forms of adaptation, however, are only dealing with changes at the level of infrastructure services, and are therefore rather elementary.

This paper introduces some benefits of using semantic technologies in the context of self-management in cloud application platforms, and sketches a possible approach to supporting self-management through the realization of an ontology-based adaptation framework.

The research work introduced in this paper is novel, because this topic has not yet been addressed in the literature, and we consider that the application of ontologies in the context of self-managing cloud platforms is a research area worth studying.

The paper is structured as follows: section 2 gives some background knowledge on cloud application platforms and self-management. Section 3 presents ontologies and benefits of utilising them in the context of self-managing cloud platforms. Section 4 is dedicated to a high-level description of an ontology-driven approach to self-management in cloud platforms. Section 5 shortly describes the related work. Finally, in section 6, we summarize the previous work and describe future research steps.

2 Background and Motivation

2.1 Cloud Application Platforms

Cloud application platforms (henceforth - cloud platforms) are transforming the processes of development, deployment and maintenance of Web-based software applications [2]. Platforms such as Google App Engine, Microsoft Azure, Force.com, Heroku, and other contemporary Platform-as-a-Service (PaaS) offerings aim at supporting the complete lifecycle of building and delivering Web applications and facilitate the deployment of software applications without the necessity of buying and managing the underlying hardware and software layers [2, 3].

Various PaaS vendor offerings differ with respect to the integrated facilities they provide to users. Some of them are just "bare" platforms with a very basic set of services. For example, Amazon Elastic Beanstalk [4] provides software developers only with an operating system, a web container for PHP or Java applications, and some basic configuration options. It is the developers' responsibility to develop and to test an application remotely, deploy it to the cloud service, and maintain it during run-time. On the other hand, there are platforms supporting the complete software development life-cycle and even liberating their users from routine tasks of developing, testing, deploying, etc. For example, with Zoho [5], users do not even need to code, as configuration and deployment of an application can be performed through a "point-and-click interface". Other PaaS offerings can provide their users with facilities and on-demand tools enabling several or all of the following features [3]:

- Application design, development, testing and deployment
- Database, storage, and persistence integration
- Bug tracking
- Team collaboration
- Application versioning
- Web service integration
- Compatibility with other cloud platforms
- State management
- Developer community support
- Auto-scaling, etc.

Regardless of the particular functionalities being offered, a common characteristic of cloud platforms is that they reduce the effort required to create a Web application, to deploy it to a hosting environment, and to maintain an underlying infrastructure [6]. Monitoring the infrastructure and ensuring that deployed applications fully function at a satisfactory level becomes a concern of the platform, rather than of developers and administrators [1]. To provide that functionality, advanced cloud platforms are equipped with mechanisms responsible for carrying out different forms of adaptation.

2.2 Cloud Platform Adaptation Mechanisms

To ensure that appropriate service levels are observed, cloud platforms are continuously monitoring the usage of deployed applications, and are capable of reacting to important changes in the environment by carrying out some form of automatic adaptation. Most commonly, cloud application platforms are capable of adapting to varying volumes and types of user requests by allocating the incoming workload across different computing resources (load balancing), or by reserving and releasing computational resources upon demand (elasticity) [2, 3]. Load balancing and elasticity are both essential properties of a cloud service, according to the National Institute of Standards and Technology (NIST) [7].

Monitoring various system characteristics during the execution of applications (e.g., response time, throughput, utilization of virtual machines and containers, etc.), and generating actions accordingly is a common mechanism in existing PaaS offerings. A mechanism of this form may provision or remove virtual machines and program licenses, allocate containers within virtual machines, launch additional threads for faster execution [8], etc.

Together with adaptive actions performed at the Infrastructure-as-a-Service (IaaS) level (e.g., provisioning and removing computational instances, network management, etc.), such a framework mainly deals with scalability and seems to be the only adaptive mechanism employed by most cloud platforms [3].

2.3 Limitations of the State of the Art in Adaptation Mechanisms

It becomes apparent that the forms of adaptation in existing cloud platforms are only dealing with changes at the level of infrastructure services, and are therefore rather elementary. More complex and intelligent adaptation scenarios, such as the need to modify the actual structure and/or behavior of an application during its runtime, are much more difficult to automate, and are presently beyond the capabilities of common cloud application platforms.

At the age of the Internet of Services, when applications are more and more depending on third-party services [9], a failure of an external service at one point may lead to malfunctioning of a whole cloud-based system, without the platform noticing it. Presently, cloud platforms are incapable of detecting and reacting to such situations. Cloud platforms cannot make a distinction between infinitely looping and long-running operations, cannot understand whether a response from an external service is correct or not, cannot substitute a web-service with another, etc. These limitations make it necessary for platform administrators and application developers to be involved in the lifecycle of an application even after it has been deployed to a

cloud environment. That is, the adaptation is done manually by rewriting the application/platform code, recompiling, redeploying the application or restarting the platform.

All these aforesaid limitations bring us to an idea of developing a *self-managing* cloud platform - what if we could enhance a cloud platform with an *autonomic* adaptation framework?

2.4 Self-management and Self-reflective Systems

According to IBM's view on autonomic computing, the following fundamental features are usually considered under the umbrella term of *self-management* [10, 11]:

- *Self-configuration*: the ability of a system to adapt automatically to dynamically changing environment
- *Self-healing*: the ability of a system to detect, diagnose and react to failures.
- *Self-optimization*: the ability of a system to efficiently maximize utilization of available resources automatically
- *Self-protection*: the ability of a system to discover and protect themselves from various malicious attacks.

One of the possible ways of achieving self-management in a cloud platform is through self-reflection. [12]. A self-reflective system refers to the use of a causally connected self-representation to support the inspection and adaptation of that system [12]. It means that such a system is self-aware of its internal structure (i.e. system subcomponents, interconnections between subcomponents, available resources, etc.) and able to perform run-time adaptations, so that applied adaptations dynamically reflect on the state of the system (thus, possibly, triggering another adaptation cycle). The motivation behind self-reflection stems from the necessity to have systems that, when deployed in hostile and/or dynamically changing settings, are capable of reacting to various changes in the environment. In such scenarios, the capability of a remote system to perform automatic adaptations at run-time, and within a specific time frame, is often of a great importance.

3 Ontologies as a Way of Achieving Self-reflection

3.1 Ontologies

Ontologies, in their contemporary sense, started attracting attention in the 1970s as a way of representing and reasoning about knowledge in the context of artificial intelligence. An ontology, as it is widely used in the modern computer science literature, is "*a formal, explicit specification of a shared conceptualization*" [13]. *Conceptualization* refers to an abstract model of a phenomenon existing in the real world. This abstract model includes only relevant concepts of that phenomenon. *Explicit* means that the type of concepts used, and the constraints on their use are explicitly defined. *Formal* means that an ontology should be unambiguous and

machine-readable. *Shared* refers to the fact that knowledge comprising an ontology is accepted and agreed on by a group of people, not just an individual [13].

By means of ontology specification languages, such as the Web Ontology Language (OWL) [14], it is possible to explicitly define knowledge (i.e. concepts, relations, properties, instances, etc.) and basic rules in order to reason about this knowledge. Rule specification languages, such as the Semantic Web Rule Language (SWRL) [14], extend ontology specification languages by providing a way of defining more complex and expressive rules. Based on explicitly defined knowledge and rules, a reasoning engine can infer additional, implicit knowledge.

Fields where ontologies are widely and successfully used are enterprise interoperability [14, 15], life sciences such as biomedicine [14], the Semantic Web [16], e-commerce [14], and more.

3.2 Potential Benefits of Using Ontologies to Support Self-reflection

Utilising ontologies to support self-reflection may be one of the possible solutions to the above-mentioned problem of ineffective adaptation at the PaaS level. An ontology-driven adaptation framework would allow platform administrators to create high-level specifications of a platform's internal structure as well its adaptation policies by means of ontologies and rules [17]. Given a set of such policies, an adaptation engine would reason, at runtime, about whether or not some adaptation needs to be performed (e.g. substitution of an external Web service) in response to particular usage conditions (e.g. the user's context, the combination of applications being used, the types of data involved, etc.).

Compared to more traditional methods explored in the autonomic computing literature [10, 11], an ontology-based approach to platform self-management could provide the following benefits for application developers and platform administrators:

- Separation of concerns: with ontologies separated from the platform/application programming code, it is easier to make changes to adaptation policies on the fly (i.e. without recompiling, redeploying and restarting the platform/application) and to maintain the adaptation framework. Moreover, this approach allows non-professional programmers (i.e. domain specialists) to develop adaptation policies as well.
- More flexible and intelligent adaptation at any level of abstraction: it is up to platform designers to decide whether adaptation actions will be carried out at the lowest level of programming classes and variables or at the upper-most level of the whole platform. It is worth mentioning here that the ontology-based approach is generic and could be applied to adaptations at the IaaS level as well. Moreover, this approach could be potentially applied not only to cloud platforms, but to other distributed systems (e.g. service-based applications).
- Increase in reuse, automation and reliability: once implemented and published on the Web, ontologies are ready to be re-used by any third parties, thus saving ontology engineers from "reinventing the wheel". Since the reasoning process is automated and performed by a reasoning engine, it is not prone to so-called "human factors" and more reliable (provided the correctness and validity of ontologies and rules).

4 Towards self-reflection in cloud-platforms

4.1 The MAPE-K Autonomic Loop

Taking into consideration all aforementioned issues, we propose to investigate an ontology-driven approach to achieve self-management in cloud application platforms. This approach implements IBM's reference model for autonomic control loops [18], known as the MAPE-K (Monitor, Analyse, Plan, Execute, Knowledge) loop. Figure 1 depicts this generic reference model.

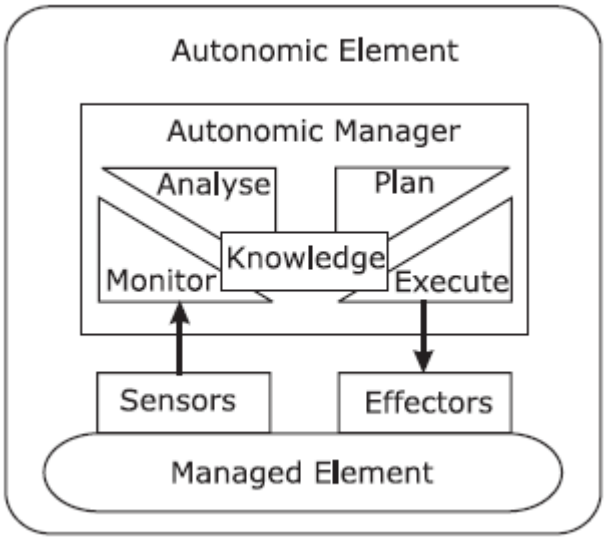

Fig. 1. IBM's MAPE-K reference model for autonomic control loops (excerpted from [18]).

Let us examine this model in the context of adaptations in a cloud platform. The *autonomic element* represents a cloud application platform, whereas the *managed element* could be either one of the deployed applications or the cloud platform itself. The managed elements are equipped with *sensors and effectors* - software components that are responsible for collecting information about a managed element and carrying out adaptation instructions to a managed element respectively.

The *autonomic manager* is a software component and the core element of the adaptation mechanism. Its main responsibilities are:

(i) *monitoring* the managed elements by means of collecting data from the sensors;
(ii) *analysing* the data collected from the sensors;
(iii) *planning* required adaptation activities based on the analysis;
(iv) *executing* the adaptation plan.

All these activities are done based on the internal (self-reflective) *knowledge* of the system. The knowledge may also include a set of policies expressed in the form of "*event-condition-action*" (ECA) rules [18]. That is, a typical policy could be represented as a following rule: "When an *event* occurs and a *condition* holds, then an *action* is executed".

Let us consider a scenario of an application system consuming a potentially malfunctioning web service. In this case, the managed element is the application system, and the result returned by the service is one of the parameters being monitored. The autonomic manager possesses knowledge about the structure of the application and its connection to the web-service, as well as the list of equivalent services that can act as substitutes. The knowledge base, among other things, also includes a policy stating that if the service returns an incorrect result, then this service needs to be substituted.

4.2 Role of Ontologies

So how could ontologies fit into this MAPE-K model? Given the characteristics of ontologies, as explained briefly in section 3.2, we believe that it is worthwhile investigating the benefits (and drawbacks) of using ontologies for representing internal knowledge of a managed system and reasoning over this knowledge.

As the main source of self-reflective knowledge comprising the knowledge base of the autonomic manager, ontologies could provide formal models of the internal structure of the application platform (e.g. its components and relations between them, concepts related to the adaptation process, etc.) and define policies for carrying out adaptation activities. This is where a major benefit of ontology-driven approach becomes obvious. We, as platform developers, are exempted from the time-consuming and effort-intensive task of implementing our own (potentially error-prone) analysis engine from scratch, because an ontology-driven approach already provides us with this functionality by means of reasoning engines. It means that a routine of reasoning over (i.e. analysing) a set of situations and adaptation alternatives, is done by an already existing, resilient, and efficient mechanism. In contrast to this approach, consider a scenario where the self-reflective knowledge would be stored in a database or XML files. In this case, we would need to "hard-code" the adaptation logic with numerous "*if-then*" and "*switch-case*" operators.

We now consider some simple examples illustrating the self-managing capabilities of an ontology-driven approach. Please note that these examples only serve the purpose of demonstrating the idea of using ontologies and rules for self-adaptation, and are therefore very simple. In practice, the set of ontologies and rules could be much more complex.

Let us consider an OWL ontology that defines a vocabulary of concepts related to the system and the adaptation process. Among other things, it includes the classes *Service*, *Time*, *System*, and *Client.* The terms *hasResponseTime, hasHighResponseTime* and *hasCrashed* correspond to data properties in the ontology, and *isEquivalent* and *needsSubstitution* are object properties whose domain is the class *Service*. The terms *isRequestedBy* is an object property of the class *System,* and *requestsPerSecond* and *isMalicious* are data properties of the class *Client*.

Let us start with a policy stating that if a response time from a web-service is greater than 5 seconds, that service can be considered to have crashed and should be substituted. This policy, represented in SWRL, could look like (1). Following these rules, the autonomic manager can deduce that a service should be substituted, thus enabling the platform to demonstrate self-healing capabilities.

```
Service(?s1) ∧ Time(?t) ∧ hasResponseTime(?s1, ?t) ∧
greaterThan(?t, 5000) → hasCrashed(?s1, true)                    (1)

hasCrashed(?s1, true) ∧ Service(?s2) ∧ isEquivalent(?s1, ?s2)
→ needsSubstitution(?s1, ?s2)
```

The following rules (2) imply that if a web-service does not reply in 2 seconds, it is considered to have a high response time and might need to be substituted. Based on this rule, the adaptation framework may optimize the system by substituting a "slow" service with a "faster" one (if there is any).

```
Service(?s1) ∧ Time(?t) ∧ hasResponseTime(?s1, ?t) ∧
greaterThan(?t, 2000) ∧ lessThan(?t, 5000) →
hasHighResponseTime(?s1, true)                                   (2)

hasHighResponseTime(?s1, true) ∧ Service(?s2) ∧
hasHighResponseTime(?s2, false) ∧ isEquivalent(?s1, ?s2) →
needsSubstitution(?s1, ?s2)
```

As an example of self-protecting capabilities, we can consider a rule defining a Denial-of-Service (DoS) attack as a case when the number of incoming external requests per second reaches 1000 (3), and these requests come from the same address. Given such a rule, the adaptation framework is able to identify potential threats and block the malicious IP address.

```
System(?s) ∧ Client(?c) ∧ isRequestedBy(?s, ?c) ∧
requestsPerSecond(?c, 1000) → isMalicious(?c, true)              (3)
```

As illustrated by these examples, the ontology-driven approach implies that ontologies should be used to not only define a common vocabulary, but also to provide the "building blocks" for constructing adaptation policies, that is, policies should be defined using ontological building blocks (i.e. concepts, relations, properties, etc.). It also becomes obvious that with the programming code and the policies separated from each other, we can benefit from much easier maintenance and evolution of the policies, because any changes to ontologies and rules do not affect the rest of the system.

4.3 Potential Challenges and Research Questions

Our research work is still at the initial stage, and presently it is hard to foresee all the challenges that will arise. Nevertheless, some important issues to be studied are already evident.

A delay between the moment when the system identifies a problem and the moment when an adaptation actually takes place, is a common problem in real-world systems [18]. In frequently changing environments, adaptation actions may become irrelevant by the time they are applied to the system, and this is an important challenge to be addressed. This problem has already been addressed in the research field of Stream Processing – processing of large, continuously flowing data streams [19]. A related research topic, *Stream Reasoning*, goes one step further and combines techniques from stream processing with ontology engineering in order to support "on the fly" reasoning over incoming streams of semantically-enriched data [20]. Stream reasoning deals with *events* – something that happens or changes in the current state of affairs [20]. In the context of self-management, an event can be a failure of a component, a response from a web-service or even a daylight saving time shift. In order to be processed, these events need to be somehow converted into computational models. However, in our case we also wish to reason about these events, and that is why ontologies could be used for representing events in stream reasoning.

Another challenge, also emerging from the area of stream reasoning, is identifying the parameters to be monitored. That is, which parameters are required to provide effective adaptation, and which parameters may be neglected? In a scenario where the managed frequently-changing system supplies the autonomic manager with bulks of data, it is essential to distinguish between the valuable information and so called "noise" [19].

This brings us to another research challenge - the minimization of the impact of the autonomic manager's presence on the overall performance of the platform. In other words, with many applications running and being monitored, the autonomic manager may become the platform's "bottleneck". We have to propose replication mechanisms (i.e. launching additional instances of the autonomic manager) and load balancing strategies for spreading the load across several instances of the autonomic manager. At this stage of the research, it is not yet clear if this will be the case, but this may be an important aspect to investigate.

Another research question concerning the future implementation of the autonomic manager is how it will interact with the platform and deployed applications within the cloud platform. That is, how the sensors and effectors will be designed and implemented. Presumably, we will need to propose an API that developers could follow when designing and implementing applications. This API would provide some entry-points for the autonomic manager to perform so called *active* monitoring [18].

5 Related Work

The research topic described so far is positioned at the intersection of three major research areas: Cloud Computing, Autonomic Computing, and Ontology Engineering. A literature survey in this direction has shown that our work seems to be novel, and apparently there has not been any research done in the direction of using ontologies in the context of autonomic cloud platforms.

Nevertheless, there have been research activities that cover two of the abovementioned research areas and provide potentially valuable insights. For

instance, much effort has been expended in order to develop autonomic clouds, although most researchers focus on achieving self-management at the IaaS level (e.g. [21–23]). Especially valuable and relevant research work was done by Maurer et al. [23], who also adapted the MAPE-K model in order to achieve autonomic management of cloud infrastructures and prevent SLA violations. Moreover, recent research by Brescovic et al. [24] has focused on self-management at the PaaS level and presented a vision of an autonomic self-aware cloud market platform. Brescovic et al. also employ the MAPE-K reference model in their work. They introduce an autonomic cloud market monitoring methodology and demonstrate its viability with relevant use-cases.

There have also been some research activities that cover the application of ontologies to autonomic computer systems. For example, research by Carey et al. [25] is relevant to our work because they have also employed ontologies to define policies, to support autonomic communications in ubiquitous computing environments. A similar approach has been described by Vassev and Hinchey [26]. In their survey the authors provide a comprehensive analysis of different approaches to self-reflective knowledge representation, and describe their experience of utilising ontologies to capture the knowledge domain and adaptation policies in the context of the Autonomic Service-Component ENSembles (ASCENS) project. Although both [25] and [26] are mainly focused on research fields distant from cloud computing, the described experience and results of using ontologies to achieve self-reflection seem to be promising.

6 Conclusions and Future Work

In this paper we gave a high-level description of an ontology-driven approach to creating a self-managing cloud platform. The approach builds on IBM's MAPE-K reference model, which is a widely used generic model for designing closed adaptation loops. Having introduced some theoretical background on the topics of autonomic computing and ontology engineering, we introduced an idea of how ontologies and rules could be utilised in order to represent the internal knowledge of the platform and provide reasoning over this knowledge, thus exempting platform developers from implementing their own analysis engine. Other benefits of using ontologies, such as the separation of concerns, increased re-use, automation and reliability, were described as well.

The future work includes a deeper study of theoretical underpinnings relevant to this project, as well as implementation details of existing adaptation engines. Particularly promising directions of the further investigation are stream reasoning and event processing, as these topics may provide several lessons to be learnt in the context of research challenges. This (yet, not exhaustive) list includes the delay in adaptation actions, the selection of parameters to be monitored, the way the autonomic manager will interact with a cloud platform, and minimizing the impact the manager's presence in the system. A survey on existing adaptation mechanisms from a perspective of their applicability to cloud platform-based scenarios may be another potential source of insights and valuable research experience.

References

1. Shroff, G.: Enterprise Cloud Computing: Technology, Architecture, Applications. Cambridge University Press (2010).
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: Above the Clouds: A Berkeley View of Cloud Computing, (2009).
3. Natis, Y.V., Knipp, E., Valdes, R., Cearley, D.W., Sholler, D.: Who's Who in Application Platforms for Cloud Computing: The Cloud Specialists. Gartner Research (2009).
4. Amazon Elastic Beanstalk, http://aws.amazon.com/elasticbeanstalk/.
5. Zoho, http://www.zoho.com/.
6. Dillon, T., Chen Wu, Chang, E.: Cloud Computing: Issues and Challenges. 2010 24th IEEE International Conference on Advanced Information Networking and Applications (AINA). pp. 27–33. IEEE (2010).
7. Mell, P., Grance, T.: The NIST definition of cloud computing. National Institute of Standards and Technology. 53, 50 (2009).
8. Litoiu, M., Woodside, M., Wong, J., Ng, J., Iszlai, G.: A business driven cloud optimization architecture. Proceedings of the 2010 ACM Symposium on Applied Computing. pp. 380–385. ACM, New York, NY, USA (2010).
9. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA (2005).
10. Ganek, A.G., Corbi, T.A.: The dawning of the autonomic computing era. IBM Systems Journal. 42, 5–18 (2003).
11. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer. 36, 41– 50 (2003).
12. Blair, G.S., Coulson, G., Grace, P.: Research directions in reflective middleware: the Lancaster experience. Proceedings of the 3rd workshop on Adaptive and reflective middleware. pp. 262–267. ACM, New York, NY, USA (2004).
13. Studer, R., Benjamins, V.R., Fensel, D.: Knowledge engineering: Principles and methods. Data & Knowledge Engineering. 25, 161–197 (1998).
14. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. CRC Press (2009).
15. Uschold, M., Gruninger, M.: Ontologies and semantics for seamless connectivity. SIGMOD Rec. 33, 58–64 (2004).
16. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American. (2001).
17. Stojanovic, L., Schneider, J., Maedche, A., Libischer, S., Studer, R., Lumpp, T., Abecker, A., Breiter, G., Dinger, J.: The role of ontologies in autonomic computing systems. IBM Systems Journal. 43, 598–616 (2004).
18. Huebscher, M.C., McCann, J.A.: A survey of autonomic computing - degrees, models, and applications. ACM Comput. Surv. 40, 1–28 (2008).
19. Barbieri, D., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: Stream Reasoning: Where We Got So Far. Proceedings of the 4th International Workshop on New Forms of Reasoning for the Semantic Web: Scalable and Dynamic (NeFoRS) (2010).

20. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. Proceedings of the 20th international conference on World wide web. pp. 635–644. ACM, New York, NY, USA (2011).
21. Martin, P., Brown, A., Powley, W., Vazquez-Poletti, J.L.: Autonomic management of elastic services in the cloud. Computers and Communications (ISCC), 2011 IEEE Symposium on. pp. 135 –140 (2011).
22. Casalicchio, E., Silvestri, L.: Architectures for autonomic service management in cloud-based systems. Computers and Communications (ISCC), 2011 IEEE Symposium on. pp. 161 –166 (2011).
23. Maurer, M., Breskovic, I., Emeakaroha, V.C., Brandic, I.: Revealing the MAPE loop for the autonomic management of Cloud infrastructures. Computers and Communications (ISCC), 2011 IEEE Symposium on. pp. 147 –152 (2011).
24. Breskovic, I., Haas, C., Caton, S., Brandic, I.: Towards Self-Awareness in Cloud Markets: A Monitoring Methodology. Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on. pp. 81 –88 (2011).
25. Carey, K., Courtenage, S., Feeney, K., Lewis, D., Tiropanis, T.: Semantic-Based Policy Engineering for Autonomic Systems. IFIP Lecture Notes in Computer Science (LNCS). 3457, 152–164 (2011).
26. Vassev, E., Hinchey, M.: Knowledge Representation and Awareness in Autonomic Service-Component Ensembles - State of the Art. Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on. pp. 110–119 (2011).