Rapid Prototyping of Architectures on the Cloud Using Semantic Resource Description

Houssam Haitof

Technische Universität München, Germany
haitof@in.tum.de

Abstract. We present in this paper a way for prototyping architectures based on the generation of service representations of resources. This generated "infrastructure" can be used to rapidly build on-demand settings for application/scenario requirements in a Cloud Computing context where such requirements can be as diverse as the applications running on the Cloud. The resources used to build the infrastructure are semantically described to capture their properties and capabilities. We have also developed a framework called the Managed Resource Framework (MRF) to automatically generate service descriptions with an added manageability interface from these semantic description. These services are then ready for deployment. Our work was materialize in the SEROM Software.

1 Introduction

One of the biggest advantages of Cloud Computing is that the back-end of the Cloud infrastructure is invisible to the user, especially for the Software as a Service (SaaS) and Platform as a Service (PaaS) types of Cloud. This advantage is less prominent when we deal with the Infrastructure as a Service (IaaS) type of Clouds where the user has more control, and hence, more visibility on the infrastructure. Another difficulty with the IaaS is that the Cloud is often preconfigured with some services suitable for an application or a certain scenario, and once the requirements change, that configuration needs to be adapted by adding or removing some services or some other infrastructure components.

Depending on the application settings, resources are instantiated to answer the requirements needed by the application. If, for instance, the requirements include some database servers and web servers, then these requirements and the relationship among them are described in a semantic document and a match-making routine searches for the appropriate resources using their semantic descriptions and, using the Managed Resource Framework, actual service representations of the resources are generated and deployed with minimum human intervention. This prototype architecture can be then reviewed and validated by an administrator if there is need for that.

For every service representation, a standard interface for managing the service is generated. This interface can then be used by other software agents for management purposes or for querying about the resource capabilities and properties. The resource that can be used in this setting have to be first semantically described following a basic model called an Abstract Managed Resource Model.

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part I, LNCS 7155, pp. 73–82, 2012.

Our architectural model is based on the Service Oriented Architecture and we use semantics with a formal language representation based on description logic for the knowledge level.

2 Description and Knowledge Levels

Our system needs a certain level of conformity and defined expectation, especially when dealing with system components. Confronting the Software components with unknown system representations where resources may take unknown forms, use unknown communication protocols or present unknown control interfaces is a real challenge that is out of the scope of this work. We believe that such system configuration is too complex to be dealt with and we think that it is extremely difficult to come up with a system capable of navigating through such a configuration. We assume that all the system component have a certain degree of conformity making it (i.e. the system) a homogeneous and integrated system. This conformity is expressed at the level of component description and at the possible ways to interact with those components. This level of predictability is necessary whenever we deal with an automated or semi-automated management model, where several management tasks are offloaded to software agents.

2.1 The Need for SOA

Having this conformity of resources at the description level coincide with the service encapsulation that the service orientation model follows. Where the service represents a single, contained and self-standing set of functionalities. In this model, everything is a service that shares similar communication protocols with all other services, allowing other software agents, services or not, to handle them in a predictable way. This important characteristic of service orientation is an appealing argument that motivated the use of that model as our conceptual model. The service orientation model is quite vague in its definitions leaving the room open for interpretation and assumptions related to many design principles. Although, it offers a lot of flexibility, this vagueness leads often to incompatible service oriented architectures defying the whole purpose off the model. This is why we wanted to stick, as much as possible, to an abstraction level that would, at the same time, adhere to the guiding principles for a service oriented system, while being extensible and compatible with the defacto standards used for creating service oriented systems and the design decision that lead to them. This, in fact, justifies some design decision such as the choice of WS*-[1] stack over REST even if the latter is easier and has a more perfo urmant implementation than the first, because the first is indeed the defacto standard used in implementing service oriented systems.

[1] WS-* is used to refer to the collection of specifications related to the Web Services.

2.2 The Need for Semantics

Even though modeling everything as a service would ease handling system components by software agents, these latter would still miss the semantic of the functionalities they are dealing with. In other terms, the software agents would have access to the "technical" description of the services and would be able to initiate and endure communication with the services, however, they will not be able to know what the service is actually doing, or to distinguish (or find, as a matter of redundancy support for instance) between different services supposedly doing the same thing. Another layer of information is needed to allow software agents to understand what they are dealing with in a manner close to what a human would do to apprehend the *usefulness* of a service. This layer is called the *knowledge layer*. But, why would the software agents need an access to the knowledge layer? In fact, it is not that difficult to automatize a system, in a majority of cases it comes down to writing a set of scripts. The real value is coming from enabling software agents to become more than automated agents and be adaptive and even more: to behave in an autonomic manner.

3 Semantic Resource Description and Resource Management

Describing the the resource characteristics, capabilities, interaction possibilities and management potential in a formal language is not enough for other software components to figure out the exact purpose of the resource and the meaning of its capabilities. Using such language provides only syntactic level description that necessitate the intervention of humans to appreciate the value or the purpose of the resource. Hence the resources need to be augmented by a semantic description that would capture the fore-mentioned meaning. A serious problem in today's IT management is the lack of information related to the managed resources [10]. This lack of information can be sensed at different levels which add to the complexity of the problem.

3.1 Lack of Information Effects

At the Level of IT System. There is a lack of information on what is installed and where it is installed. Often the purpose of the system it self is ambiguous if not unknown. Sometimes whole parts of the system that are unused are up and running and nobody can make a clear decision of the necessity of those components or if any other part of the system *may* depend on those components.

At the Level of the Resources Relationships. Often the information of the relationships of resources to one another is missing. Information about why and how a set of resources is generally not documented making failures analysis an extremely difficult task as well as problem source determination. Another

problem that is also related to the first point is that with the lack of such information, predicting the impact of a change in a system is nearly impossible, with cascading effects and latent effects being the worse types of problems that may happen.

At the Level of the Resource Itself. There is a lack of information about the resource itself, its purpose, capabilities and requirements. If this information is ever stored, it is done separately of the resource in external repositories that can get inconsistent if there is no system capable of reflecting in it the status of the resources in real-time. If such information was available, maintenance would be done in an easier manner as data about the resource is available, allowing to offer it its needed ecosystem or changing it with an equivalent resource.

3.2 Semantically Augmented Resources

Describing resources semantically is crucial to understand what they are, what they do and how to interact with them. This semantic augmentation amounts to adding meta-data to resources as descriptive layer, thus publishing information about the resources in a standardized, machine-readable way. If we want management systems in our model to interact with the resources, we need more than the capacity to how to read information about the resource (syntax parsing), but also what does the information mean (semantics).

3.3 Managing Resources

One important principle behind the creation of web services was for application integration, including legacy applications that were written using heterogeneous technologies and platforms. This was also the case for management applications that not only had to deal with heterogeneous resources, were themselves not inter-operable. Using web services principles, it is possible for management applications to use common messaging protocols between themselves and the manageable resources, thus becoming vendor-neutral and platform-independent solutions [7]. In the WS-* landscape, there are two important and competing sets of specifications that describe management messaging protocols on top of the web service stack, namely WSDM [7,20,8,9] and WS-Man [15]. There is an ongoing effort to merge the two specifications [2] as conformity and standardization was a key objective in the design of both specifications. Both specifications have a resource centric view of management, where the common messaging protocol is used to communicate directly with the resource through a standardized interface. Contrast this with the traditional model where management applications were often contacting agents on behalf of the resources. This standard interface is the interface of the web service that represents the resources. In other words, resources can be accessed only through the End Point Reference of the web service.

3.4 Representing Resources as Services

There is nothing special about representing a resource using a web service, if that resource is already offering some API to access its capabilities and properties. It would be a matter of writing an interface to this API, accessible through a well-defined web service. However there are some issues related to the intrinsic nature of web services and resources. The most prominent difference is the fact that web services are stateless, meaning that they do not keep data (a state) between different invocations. This contradicts the view of the physical resource that keeps a state during its lifetime. This stateless characteristic of web services is not a limitation as it was a design choice aiming at the web services being light-weight software components. There is mechanisms that permits to emulates a statefull behavior for web services, with the most traditional being session cookies or using persistent storage of state like a database or using WS-Session.

WSRF [3] propose an elegant and integrated solution to the stateless issue of web services by using descriptive document called *ResourceProperties* and introducing the concept of *WS-Resource*. The WSRF provides a general solution using web services to an originally specific problem: describing and representing Grid resources that are statefull in nature. Another relevant feature of WSRF is that it brings a solution for management of a resource lifetime, faults and properties. Rendering resources as WS-Resource and decomposing software component into services is the first step toward a SOA enabled management architecture with all the advantages that it can bring such as ease of management, adaptability and automation.

4 Managed Resource Framework (MRF)

This section discuses the *Managed Resource Framework (MRF)*, a framework for automatically generating ready-to-deploy service representations of resources from their semantic representations. The objective is to have a computer aided process by which resources can be rapidly *instantiated,* deployed and managed in a relatively quick and transparent manner for the user.

The framework assumes the existence of a semantic representation of a resource written in OWL that extends an Abstract Managed Resource Model and outputs a deployable service representation called *Managed Resource Archive (MRA)*. The only "human" intervention during this process would be in the case there were custom capabilities defined in the semantic representation and that lack an implementation (see figure 1).

Using the Managed Resource Framework, it is possible to generate resource artifacts that would eventually constitute the Managed Resource Archive. MRF assumes a target-based mechanism by which, only one or several constituent of the MRA could be generated as needed instead of the monolithic MRA.

The Managed Resource Archive is a deployable service representation of the resource that is generated by the Managed Resource Framework. The MRA is a

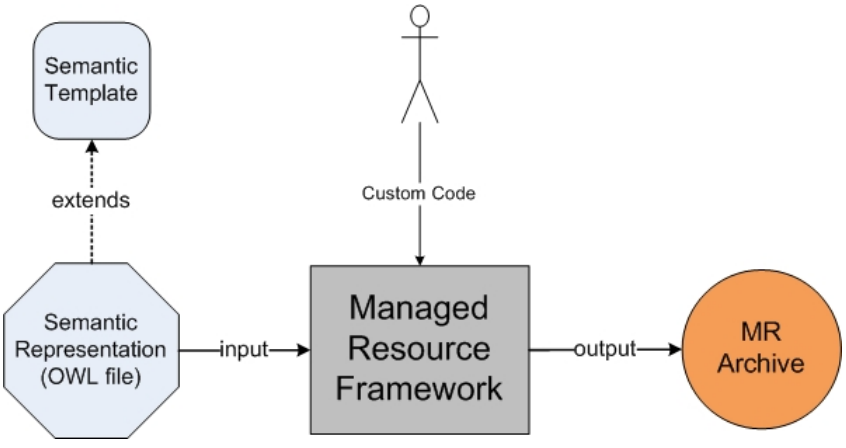

Fig. 1. Managed Resource Framework (MRF) input/output

Web application formed by a bundle of servlets, classes and other resources and intended to run on Managed Resource Containers or on Java Servlet Containers with, however, a loss of capabilities.

Once deployed, every MRA has a unique URI represented by its location in the container. An example would be `http://www.example.com/site1/res_A42`. Requests to the resource would have to start with that URL as a prefix. Every request is then forwarded to the `ServletContext` representing the resource. The `ServletContext` is an interface that defines a servlet's view of the Web application. It is up to the container provider to provide an implementation to the `ServletContext` and to ensure the one to one correspondence between every resource and a single `ServletContext`. A Web application may consist of servlets, utility Java Classes, static documents, client-side Java applets, beans, and classes and descriptive meta information.

5 Implementation

5.1 SEROM

We developed a full implementation of the Managed Resource Framework called SEROM to materialize some concepts presented in this work. The development environment was constituted of the Java programming language, eclipse and maven. We used Java for the richness of its libraries and APIs and its portability. We used eclipse as IDE and maven as project management tool.

MRF is being developed as a modular, plug-in based software to insure the future extendability of the framework. It comes in two versions a command line tool and an API to be programatically invoked. MRF is target-based, meaning that a target needs to be specified while invoking the framework and these targets can be: `model`, `wsdl`, `doc`, `source`, `proxy`, `build` or `all`. model can only

be invoked programatically, all the other targets can also be invoked from the command line tool. The `model` target generates an internal model of the resource, this model is needed by all the developed modules. `wsdl` generates the WSDL document from the model, `doc` the online documentation, `source` the server stub and the proxy skeleton source code, `proxy` the proxy client library, while `build` builds the WAR archive and finally `all` generates all of the above and packages the archive into a deployable artifact. Every new module can provide new targets but it has to specify its required inputs from the existing targets. MRF uses a templating system to generate the different documents. Appropriate templates are loaded at run-time and processed using the in memory model of the resource. The WSDM implementation used by MRF is Apache muse.

5.2 Under the Hood

MRF implementation of the management mechanisms is done following the Web Services Distributed Management (WSDM) set of standards [20,8,9]. WSDM defines protocols to allow services to expose their manageability interface in a standard way such that any consumer that is WSDM-compliant is able to access the service manageability functions. WSDM specification relies on several WS-* standards for its operations, namely WS-MetadataExchange, WS-ResourceFramework [3], WS-ResourceProperties, WS-ResourceLifetime [18], WS-ServiceGroup and WS-Notification. The manageability of a service is exposed through a Web Service and is accessed through that service EPR, called a manageability endpoint. Any software application that accesses the manageability endpoint is called manageability consumer. The manageability consumer can interact with the manageability endpoint, and hence, the resource, in three distinct ways:

- The manageability consumer can retrieve management information from the managed resource through calls to its management capabilities.
- The manageability consumer can affect the state of the managed resource by changing its state through calls to its management capabilities.
- The manageability consumer can receive notifications from the managed resource if the consumer had subscribed to receive events from the managed resource.

The methods stated above show that the relationship between the managed resource and the manageability consumer can be a pull or a pushed based communication mechanism depending on the nature of the resource and the consumer and the rate by which the resource can produce events. Producing events by the resource is, however, optional. WSDM does not, in general, define the content of the messages exchanged between the managed resource and the consumer, but only the communication protocol and the format of the exchanged data. Using MRF, the user can also specify some WSDM-defined management capabilities to be added to the resource definition. The MRF takes then care of generating the proper configuration files and capabilities implementation. The following section describes the WSDM-defined capabilities as well as other capabilities inherited from the other supporting WS-* standards.

6 Related Works

There are numerous works done in the field of semantic web services, such as WSDL-S [1] that tries to extend WSDL by adding some semantic annotation to the file as XML tags that can reference entities in models outside the WSDL document. WSDL-S builds on the establishment of WSDL as a service description language for ease of migration. WSDL-S is intended to be a minimalist approach that tries to extend the pre-existing Web Services with semantic annotation, which is quite different from the other methods that tries to create a more complete, sometimes complex frameworks. Another effort is the Web Service Modeling Language (WSMO) that tries to describe all aspects of semantic web services with the goal of automating the discovery, selection, composition, execution and other tasks related to the integration of web services. WSMO is based on the Web Service Modeling Framework (WSMF) [13] and has three working groups: WSMO [11], Web Service Modeling Language (WSML) [12] that represents a family of languages used as representation format for WSMO and Web Service Execution Environment (WSMX) [16] that acts as a reference implementation of WSMO. WSMO is composed of four elements: ontologies that define the common representation of information, web services that represent the services, goals that describes aspects of the requests to web services and finally mediators that act like connector between the different layers of WSMO. The major drawbacks of WSMO are that it is a quite complex framework that uses proprietary technologies at almost every level. It does not use WSDL, but instead a new representation formalism, it ignores UDDI for its own solution, and uses a family of languages that are not XML conform and are meant to be replacement to the already established languages such as OWL and RDF. Another work is OWL-S [14], an effort to define an ontology for semantic markup of Web Services. OWL-S was meant to be a replacement to the WSDL, however this effort was not successful. Other works on semantic web services worth mentioning here are IRS-II [17], Meteor-S [19] and SWSF [4,5,6].

7 Conclusion

The work presented in this paper aimed to bring a simple solution for rapid prototyping of architectures on the Cloud. We brought simplicity by providing a mechanism to model resource in a standard and uniform way using an expressive language for the purpose of generating software components that would provide a standard management interface to access and manage the resources represented by these components. The simplicity is apparent in modeling the resources, in the process by which the service artifacts were generated and the end-result: simple manageable components. The use of semantics allowed to capture the characteristics of the resources with simple and minimum set of descriptive classes and relationships among them and yet powerful enough to allow the conversion to other representation formats such as the service representation generated by the MRF. The use of open standards and plug-in based architecture of MRF allow for future extension of the system and ease of adaptation to other requirements.

References

1. Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M., Sheth, A., Verma, K.: Web service semantics - wsdl-s. Technical report, World Wide Web Consortium (November 2005)
2. Antony, J., et al.: Wsdm/ws-man reconciliation. Technical report, IBM (August 2006)
3. Banks, T.: Web services resource framework (wsrf) - primer v1.2. Technical report, OASIS (May 2006)
4. Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., Tabet, S.: Semantic web services framework (swsf) overview. Technical report, World Wide Web Consortium (September 2005)
5. Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., Tabet, S.: Semantic web services language (swsl). Technical report, World Wide Web Consortium (September 2005)
6. Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., Tabet, S.: Semantic web services ontology (swso). Technical report, World Wide Web Consortium (September 2005)
7. Bullard, V., Murray, B., Wilson, K.: An introduction to wsdm. Technical report, OASIS Official Committee Specification (February 2006)
8. Bullard, V., Vambenepe, W.: Web services distributed management: Management using web services (muws 1.1) part 1. Technical report, OASIS Web Services Distributed Management TC (August 2006)
9. Bullard, V., Vambenepe, W.: Web services distributed management: Management using web services (muws 1.1) part 2. Technical report, OASIS Web Services Distributed Management TC (August 2006)
10. Chess, D.M., Hanson, J.E., Pershing, J.A., White, S.R.: Prospects for simplifying itsm-based management through self-managing resources. IBM Systems Journal 46(3), 599–608 (2007)
11. de Bruijn, J., Bussler, C., Domingue, J., Fensel, D., Hepp, M., Kifer, M., Knig-Ries, B., Kopecky, J., Lara, R., Oren, E., Polleres, A., Scicluna, J., Stollberg, M.: Web service modeling ontology (wsmo). Technical report, WSMO (April 2005)
12. de Bruijn, J., Lausen, H., Krummenacher, R., Polleres, A., Predoiu, L., Kifer, M., Fensel, D.: The web service modeling language (wsml). Technical report, WSMO (October 2005)
13. Fensel, D., Bussler, C.: The web service modeling framework (wsmf). Technical report, Vrije Universiteit Amsterdam (2002)
14. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: Owl-s: Semantic markup for web services. Technical report, White Paper, DARPA Agent Markup Language (DAML) Program
15. McCollum, R., Murray, B., Reistad, B.: Web services for management (ws-management) specification. Technical report, DMTF (2008)
16. Moran, M., Zaremba, M.: Wsmx architecture. Technical report, WSMO (June 2004)
17. Motta, E., Domingue, J., Cabral, L., Gaspari, M.: Irs-ii: A framework and infrastructure for semantic web services. Technical report, Knowledge Media Institute at the Open University, Milton Keynes, UK

18. Srinivasan, L., Banks, T.: Web services resource lifetime 1.2 (ws-resourcelifetime). Technical report, OASIS (April 2006)
19. Verma, K., Gomadam, K., Sheth, A.P., Miller, J.A., Wu, Z.: The meteor-s approach for configuring and executing dynamic web processes. Technical report, LSDIS Lab, University of Georgia (June 2005)
20. Wilson, K., Sedukhin, I.: Web services distributed management: Management of web services (wsdm-mows) 1.1. Technical report, OASIS Web Services Distributed Management TC (August 2006)