Journal of Mobile, Embedded and Distributed Systems, vol. IV, no. 2, 2012
ISSN 2067 – 4074

Agent-Based Framework for Implementing and Deploying of SOA

Alexandru BUTOI, Gabriela Andreea MORAR, Andreea ILEA
Business Information Systems
Babes-Bolyai University
Cluj-Napoca
ROMANIA
alexandru.butoi@econ.ubbcluj.ro, gabriela.morar@econ.ubbcluj.ro,
andreea.ilea@econ.ubbcluj.ro

Abstract: In distributed organizational and business information systems' contexts, Service-Oriented Architectures (SOA) provide standard-based and protocol independent solutions. Despite the advances in SOA models and design methodologies, the implementation and deployment of service choreographies are still made in an un-unified manner using the existing tools. We present a three-layered framework model based on deployment agents, which allows designing and implementing service choreographies in a unified and reusable manner using the Object-Oriented Paradigm concepts. Deployment agents contain automatic mechanisms for service deployment which will ease the management of service references involved in service compositions at development time.

Key-Words: SOA, SOAP, REST, SoaML, OOP, Deployment Agents, SOA choreographies

1. Introduction

Nowadays, web services are one of the main issues in computer science and business information systems. They provide a powerful instrument to collect capabilities and components in a unified interface [5]. W3C defines the web service as a "software system designed to support interoperable machine-to-machine interaction over a network." Web services can integrate isolated resources and software systems and make them communicate in distributed environments. This integration triggered the emergence of web services open standards that have been defined based on the existing protocols like HTTP and XML and lead to the introduction of SOA. These architectures provide loosely coupled, standard-based and protocol independent solutions in distributed environments [4].
[2] defines SOA in an organizational context as a "group of components that work together for delivering a coherent set of activities of intangible nature that provide added value for an organizational user in an organizational network". Nevertheless, in the context of business information systems, these architectures can provide powerful instruments for fine-grained modeling of the intra- and inter-organizational business processes compared to a classical software system which is more rigid.
In our proposal we consider the generic model developed by [8] which describes the structural and functional components of a SOA: *service-components* (characterized by granularity, reusability and location transparency), *contracts and interfaces* (describe the nature of service and controls the service access), *service container* (execution environment), *connectors* (message exchange, interoperability requirements) and *service discovery* (mechanisms and the spatial aspect of the architecture).
We identify three main discussion contexts when talking about SOA: the *organizational* context (the applicability in business processes), the *technological* context (protocols, communication infrastructure, implementation technologies) and the *general* context (resource access over the Internet, standardization, public or private information delivery). Our work will focus on the technological context of these systems.

Moreover, from the technical point of view, we will follow the two main directions concerning the web service complaints and standards: SOAP (Simple Object Access Protocol) and REST (Representational State Transfer).

Table 1 SOAP vs. REST [3]

SOAP
Protocol independent
Models action sets accessible through single URI
Proposes a technical specification set for web service interaction
Limited addressing using POST
Accurate description of functionalities using WSDL
Interoperable framework based on Remote Procedure Call (RPC).
REST
Based on HTTP Protocol
Models resources accessible through single URI
Proposes a constraint-driven approach for web service design
High level of generalization and reusability, it uses HTTP verbs for interactions: GET, POST, PUT, DELETE
Resource description is limited to Content-type specification in HTTP header.
Predefined verbs for interaction provides higher interoperability

The latest application of web services and service composition is identified in Software as a Service where cloud-based application can gain access to cloud resources trough standardized interfaces provided by WS Standards and cloud resources can be aggregated using service compositions and service choreographies [9].

2. Modeling SOA with SoaML

Literature already handles modeling SOA with SoaML (Service Oriented Architecture Modeling Language), which is meant to support the activities of service modeling and design to fit into an overall model-driven development approach [1]. SoaML specification [7] is an OMG specification for modeling service oriented architecture and meets the mandatory requirements of the UPMS (Unified Modeling Language Profile and Meta-model for Services) and extends UML 2.0 in order to allow the specification and design of services within a service oriented architecture. These extensions support the specification of systems of services, the specification of individual service interfaces and the specification of service implementations. Moreover, they offer new modeling capabilities [7] such as: identifying services and the requirements they are intended to fulfill, defining service consumers and providers, how they are connected and how the service functional capabilities are used by consumers and implemented by providers in a manner that is consistent with both the service specification protocols and fulfilled requirements.

Some of the basic concepts of SoaML were implemented in [6] as an Ecore model using the Eclipse Modeling Framework (EMF), which is a tool for modeling SOA using SoaML and generating OSGi Declarative Services Models from SoaML models. The SoaML concepts considered for this implementation, as presented in [6] and [1] are the following:

1) Service interfaces which describe the operations used between a service provider and a service consumer from the perspective of the provider.

2) Service contracts are used to define the terms, conditions, interfaces and choreographies that participants must agree on. They are used to specify services that are provided and consumed based on interactions of the participants.

3) Participants allow defining the service providers and consumers.

4) Capabilities represent an abstraction of the ability to identify or specify a set of functions or resources that a service provided by the participants might offer.

5) Participant architectures represent a high level view of a SOA defining how a set of participants work together in order to provide and use services.

6) Service channels provide a communication path between consumer requests and services provider.

ISSN 2067 – 4074

7) Service data are used to describe service messages and message attachments that are exchanged between service consumers and providers.
Moreover, according to [7], SoaML has been designed to support both an IT and business perspective of SOA. Elvesaeter et al. imply in [1] that a clearer separation of the business-level and IT-level concepts are needed and they suggest integrating the use of the SoaML language with the Business Motivation Model (BMM) language to define business motivation models in order to align process models with service models for SOA.

3. The proposed framework-prototype

Els van de Kar in [2] and several other authors proposed some approaches of modeling service-oriented architectures in a coherent and unified manner, taking into account the Quality of Service (QoS) and consumer's requirements. On the other hand the implementation of a complex system in SOA architectures is still an incoherent process and uses instruments which provide limited reusability of the code and limited debugging techniques.
We propose a conceptual framework prototype for the modeling and implementation of service choreographies which will enable the designer and developer to make use of the Object-Oriented-Paradigm in order to easily implement such choreographies. The implementation process should be as easy as implementing a usual software system using classes and object hierarchies.

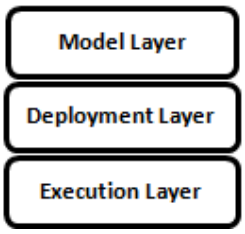

Figure 1. Framework prototype overview

The Model Layer corresponds to the implementation environment where the developer can define the class hierarchy and the implementation of the SOA system. This layer is characterized by a programming environment with a corresponding programming language in which the developer can implement the class structure.
The Deployment Layer provides a middleware solution for deploying the implemented web services and for integrating already deployed or existing web services into the model.
The Execution Layer provides the necessary resources for the deployed web services to run at the container level.
We started from the key concepts that characterize SOA components presented in **Error! Reference source not found.**1. e modeled the web service container as a collection of services which is deployed at a well determined location and each web service is addressable through an URI. Web services are regarded as class objects that provide the maximum granularity of the components and maximize the reusability of the web services. Data structures handle the input, output or intermediary results of the web service methods while the methods implement the logic of the data transformation processes according to the requirements of the system. Each web service exposes its functionalities through interfaces or data contracts which are published in a standardized form given by the protocol specification (SOAP or REST) manner.

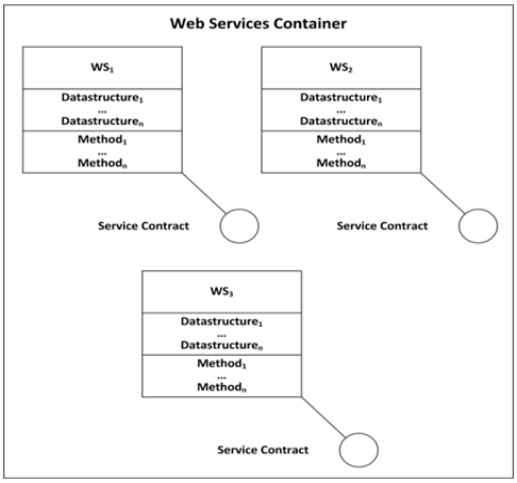

Figure 2. SOA Key Concepts Modeling

At the Model Layer level we define a class structure, which allows implementing both WS protocols in a unified and transparent

way, as it can be seen in **Error! Reference source not found.**. At the top of the hierarchy, we defined the abstract class Protocol that provides an entry point for the execution of an existing web service in the container. The SOAP and REST classes are inherited from the Protocol abstract class and implement the behavior of each protocol through methods which model the connection processes and messages exchanged according to the protocol specification. Implementation of a new SOAP web service in the container is done by extending the SOAP class and adding the business logic to it. Implementation of a new REST service is done in a similar way.

Each web service is modeled as a class that encapsulates functionalities in the form of methods and data structures. In the proposed conceptual model, we redefined the relationships between the classes in order to make Object-Oriented Paradigm to fit for SOA modeling as well:

1) Inheritance/generalization involves the membership to a specific protocol. For example in **Error! Reference source not ound.**, WebService1 implements a SOAP service and WebService2 class implements a REST service. These classes cannot be inherited anymore, but they can be reused by aggregation or composition.

2) Aggregation describes an asynchronous invoke of a service and it is used when the output data from the invoked service is not mandatory for continuing the invoker's processes. For example, a simple client application makes an asynchronous call for WebService1 and after the invoked call is sent, the application does not have to wait for the response in order to continue its processes.

3) Composition describes a synchronous invoke of a service and it is used when the output data of the service it is mandatory for continuing the invoker's processes. For example in **Error! Reference source not ound.**, WebService1 from Container 2 makes a synchronous call to WebService2 from Container 1 and has to wait for the response in order to continue its started tasks.

The class hierarchy provides a single execution entry point for each container and based on the URI list from the protocol entry class, it maps the execution of the invoked service accordingly.

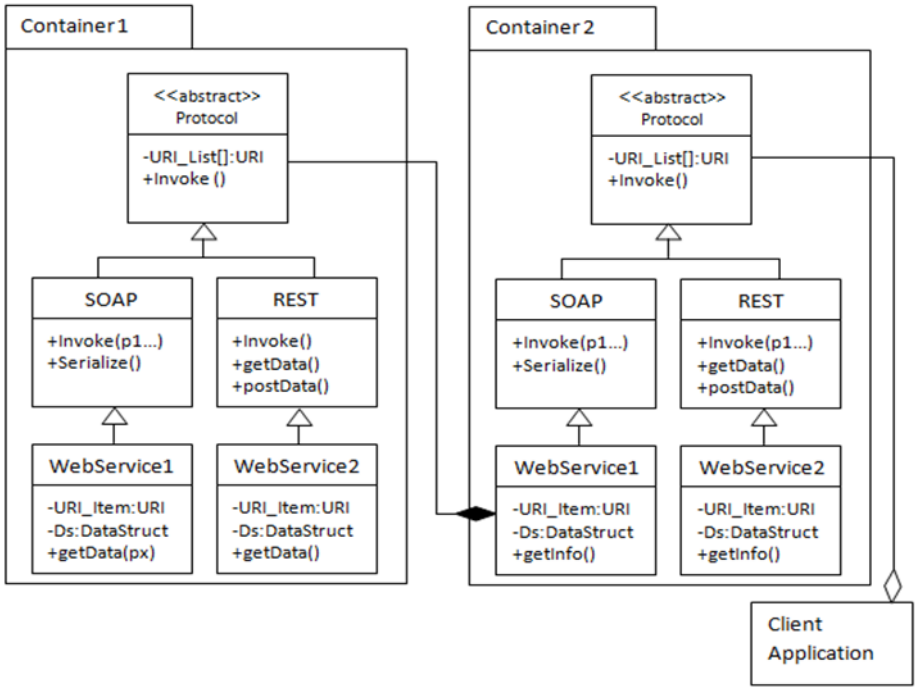

Figure 3. Web Services Choreography Modeling using OOP

Going further to the Deployment Layer (Figure 4) the framework must be equipped with automatic deployment mechanisms which will transform the

ISSN 2067 – 4074

model in a service-oriented system. Debugging Agent verifies the model for consistency and other errors and sends notifications accordingly. If the model is consistent and error free it is parsed further to the deployment agents who migrates the service components into the execution environments. SOAP Deployment Agent is responsible for the deployment of the SOAP based service components modeled at the bottom layer. The REST Deployment Agent is responsible for deploying the REST based services. The Service Composition Agent maps the designed service compositions into the execution environment. All agents from the deployment layer communicate and coordinate each other as follows:

1) Debugging Agent analyzes the model for errors or inconsistencies and sends feedback to the deployment agents whether the model is suitable or not for the next deployment phase;

2) Service Composition Agent communicates with the SOAP and REST Deployment Agents and sends information regarding the identified compositions, service references and URIs of each deployed service in order to correctly build the service choreographies.

Deployment Agents deploy each web service component together with its composition relations in a web location at well determined URIs. All web locations are considered in our framework as part of the execution layer. When the system is at the development phase of its life cycle, the web locations are instances of a development server which runs on localhost.

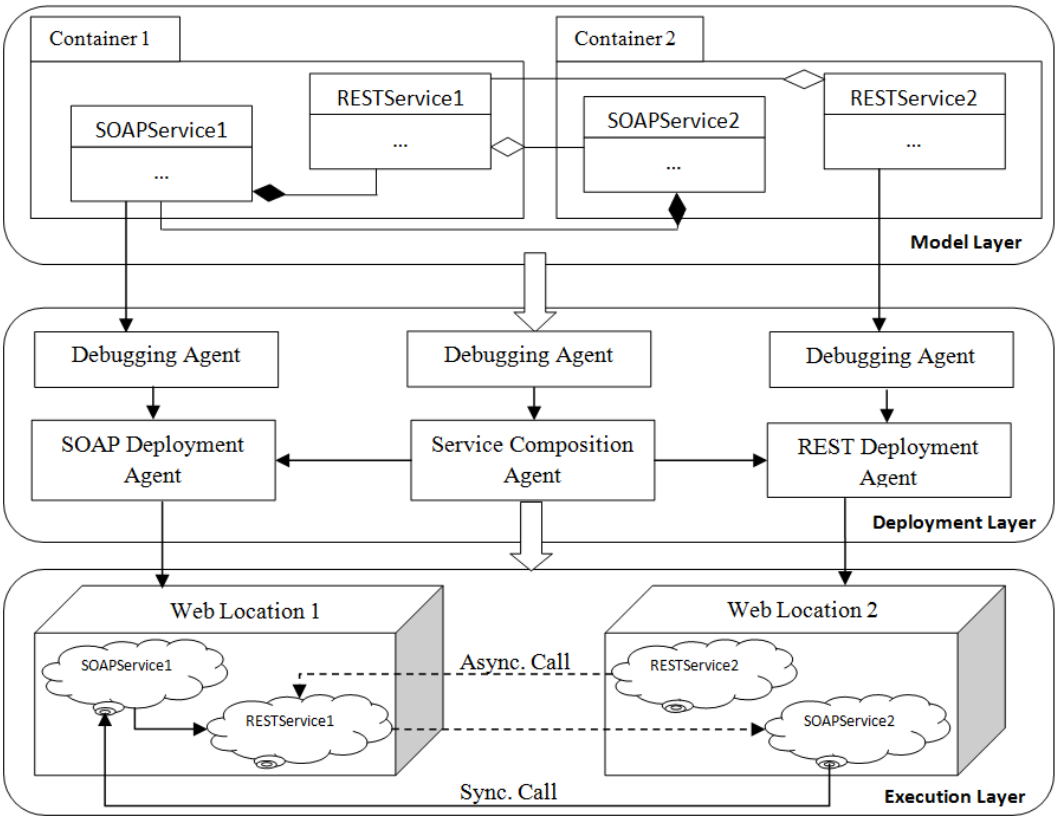

Figure 4. Deployment model of service choreographies

When the system is in the release phase of its cycle, the web locations are the deployment servers or environments where the services will be installed and available.

We observe how the service deployment agents transform the classes from the

model layer into service components at a well determined location in the Execution Layer and the service composition agent maps the defined relationships between model classes into web service calls (asynchronous or synchronous) at the execution layer.

4. Discussion

The prototype tackles the problems raised by the lack of adequate tools for SOA modeling and implementation. For example if we look at Microsoft's Windows Communication Foundation which is a framework for developing such systems, we will find some difficulties in debugging and deploying the implemented services and the services are implemented rather in an isolated manner and not in an unified and coherent manner. This can lead to difficulties in implementing the service composition due to static management of the service references and URIs. The programmer has to take care that the URIs of the service references are correct and has to be aware of changes in service references which can be very difficult in case of very large and complex systems.
Our prototype tries to solve these issues and offers a unified environment for designing and implementing service systems using object-oriented programming paradigm. It is a framework suitable for Rapid Application Development of SOA systems and prototyping involving less planning and increased reusability of the components. We tried to remove the problem of the static management of URIs by delegating the responsibility of managing the service references to the Service Composition Agent in collaboration with the service deployment agents which will ease the process of implementation and deployment of service compositions and choreographies. The programmers will con-centrate only on implementation of the class structure which models the system because the deployment will be automatically made by the agents from the deployment layer.

Our framework has some limitations as well: it does not cover the aspects of data contracts, authentication and authorization of the access which in Windows Communication Foundation are defined more clearly and it does not cover the aspects regarding to message interactions in service systems.

5. Conclusions

In this paper we proposed a conceptual framework prototype that offers a unified environment for the modeling, implemen-tation, debugging and automatic deploy-ment of service-oriented architectures based on SOAP and REST services. The framework allows designing and implemen-ting service choreographies and service compositions using class hierarchies, class composition and aggregation. It provides automatic mechanisms managed by the deployment agents who are responsible for: the deployment of the components at a certain location, for the management of the service references involved by the service compositions and none the less for consistency validation of the model. All these will ease the development of such systems by using the Object-Oriented-Paradigm and by providing high reusability of the components and location transparent deployment. Future research efforts will be focused on fine-tuning the model by applying it for real SOA implementations, evaluating the QoS for these systems and developing optimization models in service-oriented architectures based on this prototype.

Acknowledgments

G. Morar acknowledges support from the: Investing in people! Ph.D. scholarship, Project cofinanced by the Sectoral Operational Program for Human Resources Development 2007 – 2013, contract nr. POSDRU/88/1.5/S/60185 – "Innovative Doctoral Studies in Knowledge Based Society"
A. Ilea and A. Butoi acknowledge support from the: Performance Research Scholarships for Students 2011-2012,

ISSN 2067 – 4074

contract nr. 30068/ 36/19.01.2012 and contract nr. 30068/ 37/19.01.2012
This paper was supported by CNCS TE 316 Grant.
Parts of this research have been published in the Proceedings of the 11th International Conference on Informatics in Economy, IE 2012

References

[1] Brian Elvesaeter, Dima Panfilenko, Sven Jacobi, and Christian Hahn, Aligning business and IT models in service-oriented architectures using BPMN and SoaML, *In Proceedings of the First International Workshop on Model-Driven Interoperability* (MDI '10), 2010

[2] Els van De Kar and A. Verbraeck, ,Designing Mobile Service Systems, *IOS Press*, Amsterdam, The Netherlands, 2007.

[3] R. T. Fielding, *Architectural Styles and the Design of Network-Based Software Architectures, Ph.D. Dissertation.* University of California, IRVINE, 2000

[4] M. P. Papazoglou and Willem-Jan Heuvel, Service oriented architectures: Approaches, technologies and research issues, *The VLDB Journal 16*, July 2007, pp. 389-415.

[5] Mojtaba Khezrian, Wan M. N. Wan Kadir, Suhaimi Ibrahim, Keyvan Mohebbi, Kanmani Munusamy, and Sayed Gholam Hassan Tabatabaei, An evaluation of state-of-the-art approaches for web service selection, *In Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services (iiWAS '10)*, 2010, pp. 885-889.

[6] Nour Ali, Rukmani Nellipaiappan, Rajalaxmi Chandran, and Muhammad Ali Babar, Model driven support for the Service Oriented Architecture modeling language, *In Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems*, 2010, pp. 8-14. [7]http://www.omgwiki.org/SoaML/doku.php?id=specification, Retrieved June 1, 2012

[8] Karl Rehrl, Manfred Bortenschlager, Siegfried Reich, Harald Rieser, Rupert Westenthaler, Towards a Service-Oriented Architecture for Mobile Information Systems, *In: IFIP TC8 Working Conference on Mobile Information Systems (MOBIS),* Oslo, Norway, 15-17 September 2004

[9] William Voorsluys, J. B., Introduction to Cloud Computing, In J. B. R. Buyya, *Cloud Computing: Principles and Paradigms*, New York: Wiley Press, 2011, pp. 1-41