International Journal of Networked and Distributed Computing, Vol. 3, No. 3 (August 2015), 137-149

Towards a Monitoring Framework for the Automatic Integration of the Access Control Policies for Web Services

Mohammed Alodib
Department of Management Information System, Qassim University
Buraidah 51411
Qassim, Saudi Arabia
E-mail: alodib@qu.edu.sa

Abstract

Cloud computing is an emerging model of business computing that can be used to maintain configurable resources such as processors, networks, storage, operating systems, databases, and servers. Service oriented Architecture (SoA) is one of most successful paradigms deployed over the Cloud; is a layered architecture to organize software resources as services and to facilitate their deployment, discovery, and combination to produce new services. The coordination of innovations carried out between these services is a key challenge of SoA, as any failings in such services could result in a lack of availability, which may violate Service Level Agreements, leading to financial penalties or customer dissatisfaction. A service known as the Protocol service can be introduced and integrated within a system to coordinate innovations. This service may then be linked with a service called the Access Control Policies (AC_Policies) service, which identifies permissions of invocation of the Protocol service. Next, a real-time Business Activity Monitoring (BAM) dashboard is automatically generated to supervise the process of assigning the Access Control Policies, along with the status of the executions of Web services. This approach is achieved by harnessing the capability of the Model-Driven Architecture (MDA) to facilitate the automatic generation of services. As a proof of concept, this approach is implemented as a plugin.

Keywords: Web Services, Model Driven Architecture, Access Control Policies, Integration, Monitoring

1. Introduction

Cloud computing, an emerging technology that has gained popularity swiftly, is based on the sharing of resources and infrastructure [22]. Service-oriented Architecture (SoA) is a paradigm that offers a layered architecture to organize software resources as services; they can be deployed, discovered, and combined to produce new services [24]. In this type of architecture, a service can be executed using remote services or applications irrespective of their programming language, operating system or hardware platform. Coordinating of access control policies for these services is an important task that aims to ensure that individuals and applications only invoke services they are permitted to access. Therefore, an approach to automating of the assignment of access control policies is proposed.

In [4], a model-driven approach was introduced to automate the coordination of invocations of Web services, based on the dynamic discovery of the best available services, and involving the ranking of Web services using a history of invocations. In particular, the method aims to discover the most suitable service to avoid excessive use of services, thereby, enhancing system performance. This objective is achieved by introducing a new service called a Protocol service, which coordinates and controls all interactions. The Protocol service is initiated by requests from consumers; it forwards requests to the target service, obtaining the results from the provider, and returning the results to the consumer.

In [3], the approach was extended to consider the assignment of Access Control Policies (AC_Policies). This extension is based on the automatic generation of an

Published by Atlantis Press

AC_Policies service, which can act as an engine to check invocation policies. This service is integrated with the Protocol service. Consequently, prior to executing the target service, the Protocol service should interact with the AC_Policies service to identify whether the invocation is permitted.

The aim of this paper is to extend the approach to produce a method of monitoring the assignment of Access Control Policies and the execution of Web services. More specifically, the method is designed to provide a dynamic analysis of the Access Control Policies by automatically generating of a Business Active Monitoring (BAM) service, which provides a complete solution for building interactive, real-time dashboards and proactive alerts for monitoring business processes [37, 41]. Such dashboards are based on the visualization of data as it relates to invocations and the assignment of access control policies, and thus, they help managers make better decisions. They also allow users to gauge the impact on key performance indicators, which affects their business and allow them to take remediable action for recovery when necessary [37, 41].

The approach relies on harnessing the capability of Model Driven Architecture (MDA) to automate the creation of a BAM service. The BAM service is linked to the Protocol services, which are generated in a distributive manner and integrated into the system along with the AC_Policies service. This ensures that each site has its own Protocol service, AC_Policies service, and BAM service. The approach is implemented as a plugin.

This paper is organized in the following manner: Section 2.2 presents a brief review of SoA; Section 2.3 reviews the Web services; Section 2.4 presents the principles of MDA; Section 3 describes the problem; Section 4 presents the solution, implemented as a plugin; and, finally, Section 5 describes the implementation of the approach.

2. Preliminaries

2.1. *Cloud Computing*

Cloud computing is one of the most rapidly emerging technologies for sharing resources. It consists of a shared pool of configurable resources, including processors, networks, storage, operating systems, databases, and servers [22]. The main form of cloud computing is the virtualization of such resources, which can also be provided as on-demand Web services, accessed remotely over a network using a simple cloud interface. These Web services specify the following essential characteristics of cloud computing [12]:

- On-demand self-service: services can be provided to consumer unilaterally and on demand, without the need for human interaction.
- Broad network access: services are available over the network in real time through standard mechanisms.
- Resource pooling: resources are pooled to enable the provision of parallel services to multiple users (a multi-tenant model) and are adjusted to the specific demands of each user.
- Elasticity: resources are provisioned rapidly in various fine-grained quantities, enabling systems to be scaled as required. To the customer, the resources appear to be unlimited.
- Measured quality of service: the services leverage quantitative and qualitative metering capabilities, enabling usage-based billing and validation of service quality.

The architecture of Cloud computing is designed as a layered stack [12]; the Infrastructure-as-a-Service (IaaS) layer is used to describe the hardware (e.g., computers, mass storage systems, networks, databases, etc.). Developers primarily use the Platform as a Service (PaaS) layer as the Programming Environment (PE), as well as an Execution Environment (EE) such as Django Framework [20] or Sun Caroline [15]. This layer also includes operating systems and the services required for applications. The Software as a Service (SaaS) layer enables customers to run applications remotely from the Cloud (i.e., it is unnecessary to install software locally.). The data-Storage-as-a-Service (dSaaS) layer involves storage that can be used by the consumer. The Humans-as-a-Service (HuaaS) layer demonstrates that the Cloud paradigm can be extended to include services provided by humans functioning as resources.

2.2. *Service oriented Architecture (SoA)*

Component-based systems based on traditional software architecture have been developed over the past 40 years. Recently, this effort has involved building distributed components over the middleware in order to provide integrated systems, while processes have become a key demand of the IT industry. In order to meet this demand, a set of platforms have proposed, including CORBA [32],

DCOM [38, 23], and Java RMI [18]. However, such platforms face issues of interoperability as some are platform dependent, while others are programming language dependent, such as Java RMI, which only supports a single programming language (i.e. applications interacting with each other with the help of RMI must be programmed by Java). These drawbacks have led to a lack of heterogeneity and interoperability when developing integrated and distributed systems [17]. SoA has been introduced to address those challenges.

SoA is directed toward the implementation of business processes via the composition of interactive services [24]. A simple SoA infrastructure is made up of three independent collaborative components [35, 27] (see Figure 1):

- The service provider: The owner of the services (company or organization) is responsible for publishing the services.
- The service requester: a client or organization wishing to use a service. The requester searches the service registry for the desired Web services.
- The service registry: A global registry acts as a central service, providing a directory in which service descriptions are published by the service provider. Service requesters locate service descriptions in the registry and obtain binding information regarding these services from the service provider.

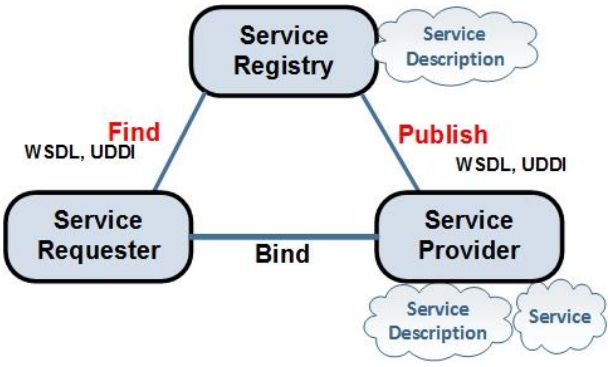

Fig. 1. Basic Service-oriented Architecture [27].

2.3. *Web Services*

Major A Web service is a component of middleware technology providing integrated and interoperable interactions over the Internet [13]. Web services offer a preferred solution to the issue of integration among autonomous and heterogeneous software systems [29]. They are well-defined, self-contained, loosely coupled, self-describing, modular applications that can be published, located and invoked across the Web network [29]. Such features ensure that Web services can be invoked dynamically by alternative applications (or Web services), and they are typically composed in tandem with other services in order to complete complex tasks. Thus, Web services are highly reusable components that act as building blocks to develop service composition and to resolve issues of application communication and integration.

The development of a composite Web service is built on the SoA paradigm [6]. Communication regarding the composition of Web services is based on the use of well-accepted standards and the Extensible Markup Language (XML) messaging framework [28]. Such standards can be used to encapsulate the service's business logic and functionality, in order to expose the functionality (but not the implementations) arising via the accessible interfaces. Thus, application programs communicate with one another, regardless of their programming language, operating systems, or hardware platforms.

Web services communicate through means of a common XML, XML Schema Definition (XSD) [40], and standard TCP/IP-based communication protocols. Moreover, a number of XML-based standards are used by Web services to describe their architecture, along with their capacity for intercommunication, collaboration, and discovery [29]. In particular, communication messages taking place between a service requester and a service provider are encoded into *Simple Object Access Protocol (SOAP)* messages (i.e., plain text XML messages). The *Web Services Description Language (WSDL)* is used to describe the invocation details of a Web service, including the service name, the available operations, and information related to the input and output variables. *Universal Description Discovery and Integration (UDDI)* provides certain protocols for querying and updating Web service information. Web services utilize existing standard TCP/IP protocols for communication purposes, including HTTP, HTTPS, SMTP, and FTP [36].

2.4. *Model Driven Architecture (MDA)*

Model Driven Architecture (MDA) [21, 26] is a framework introduced by the Object Management Group (OMG) in order to promote the role of modeling in

software development. One of the main goals of MDA is model transformation, a process whereby models in a source language are mapped in order to facilitate capture in the destination language. In the context of MDA, model transformation is defined by a number of transformation rules that specify the mapping of the *meta-elements* of the constructs of the *metamodel* of *the source language* in to the meta-elements of *the destination language*. The metamodels of the *source* and the target language are specified using a common language, known as the Meta Object Facility (MOF) [30]. In general, models in MDA are instances of metamodels. Figure 2 shows that an MDA transformation is defined from the source metamodel to the destination metamodel. Subsequently, each model identified as an instance of a corresponding metamodel is transformed automatically to an instance of the target metamodel.

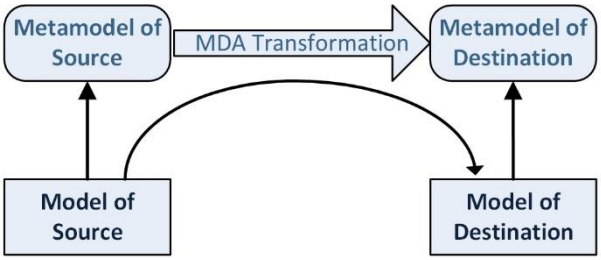

Fig. 2. Model-Driven Architecture (MDA) overview

The Meta Object Facility (MOF) Query/View/-Transformation Specification (QVT) [33] is the OMG specification, which is proposed as a method to specify model transformation rules with MOF. QVT provides a declarative and imperative language, structured into a layered architecture consisting of Relations, Core, and Operational Mappings: (1) Relations language is a high-level language that provides a textual and graphical notation for defining the mappings; (2) Core language is a specialist language based on Essential MOF and Object Constraint Language (OCL) that is used to support pattern matching and evaluating conditions; and (3) QVT Operational Mapping language is a high-level imperative language that acts to extend OCL [34] by facilitating features that are essential for writing complex transformation rules (e.g., the ability to define loops) [33]. QVT Operational Mapping language is used in the current study to obtain the specifications for the transformation rules.

QVT Operational Mapping language, specified as a standard method for providing imperative implementations, is based on the use of MOF as a repository for metamodels. The general syntax for the body of Operational Mapping is depicted in Figure 3, where the *source* is the source of the model transformation. The *mappingFunc* is the name of the model transformation, which may require some input, as captured by variable *parms*. The *target* is the destination model of the transformation. The *"init"* part has some code, which can be executed prior to implementing the main body of the mapping rules. The *population* is then used to populate the results of the mapping. The code included in the *end* part is executed before the operation is completed. The *"when"* part has a Boolean *expression* that needs to be verified as true before execution commences. The *"where"* part includes the conditions that required to be satisfied by the model elements involved in the mapping (i.e., it acts as a post condition for the mapping operation).

```
mapping source::mappingFunc(parms):target
when {...}
where {...}
{
init{...}
population{...}
end{...}
}
```

Fig. 3. The general syntax for the body of a mapping operation

There are a large number of industrial and academic case tools supporting Model Transformations, including Kermeta [25], Arcstyler [8], ATLAS [31] and Simple Transformer (SiTra) [2]. In this paper, SiTra [2] transformation engine is used to execute the transformation rules. SiTra is a lightweight Model Transformation Framework using Java for writing Model Transformations and for providing a minimal environment for transformation execution. Figure 4 depicts an overview of SiTra, which consists of two interfaces: (1) the Rule interface, in which user defined mapping rules are necessary, and (2) the Transformer interface, which provides the skeleton of the methods used to carry out the transformation. The modeler is needed only to define the transformation rules by implementing the Rule interface, which consists of three methods: *check()*, *transform()*, and *setProperties()*. If the rule is applicable for the source element in question, the *check()* method of the rule implementation is returned as true, and the *build()* method is executed. The *build()* method generates the target model element. The

setProperties() function is then used to set the attributes and links for the newly created target element. SiTra has been successfully applied to Model Transformation in various application domains [42, 14, 7, 5].

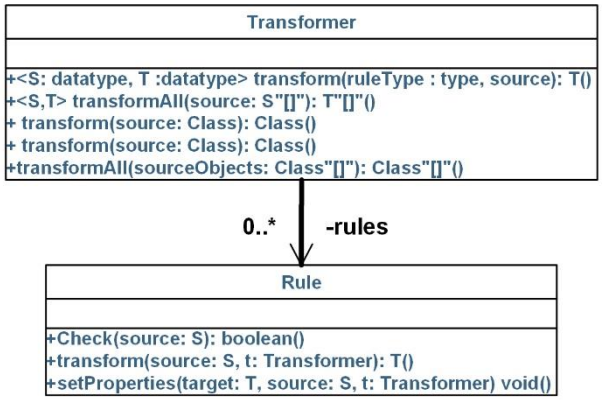

Fig. 4. Overview of SiTra

2.5. *Oracle Business Activity Monitoring (BAM)*

Oracle Business Activity Monitoring (BAM) is a framework developed as a tool to visualize data on a real-time dashboard for monitoring purposes [37, 41]. Such a promised modeling tool also provides alert facilities. Different charts (e.g., line, bar, 3D bar, 3D line, 3D area, 3D combo, 3D pie) can be used to represent the data on the dashboard, offering the following benefits:

- Event capture and data collection from composite sensors.
- Correlation of data, metrics, and key performance indicators (KPIs).
- Real-time dashboards and alerts.
- Java Message Service (JMS) messaging and web service interfaces

The BAM framework enables users to create real-time dashboards that visualize important data received from business processes. In order to build a BAM dashboard, it is necessary to create a Data Object on which to store data related to activities that required monitoring. An activity sensor (i.e., a Business Process Execution Language (BPEL) component that can be integrated and linked with a BPEL activity in order to send data to the BAM server) can then be added to a BPEL activity the user intends to monitor. Such sensors are responsible for sending information to the JMS and database providers that store the information. Sensors are classified into three types [37, 41]:

- Activity sensors, which can be used to monitor the execution of activities
- Variable sensors, which can be used to monitor BPEL process variables (e.g., input and output data of a BPEL process)
- Fault sensors, which are used to monitor BPEL thrown faults

These sensors need to be associated with a BAM Sensor Action responsible for sending the required data to a specific publisher. In this context, a publisher can be 1) a database; 2) a JMS Queue; 3) a JMS Adapter; or 4) a BAM adapter. Additional information related to Oracle BAM can be found in [37, 41].

3. Description of the Problem

The nature of SoA is to support an interoperable service-to-service interaction allowing the provision of an architecture in which each service is able to interact with other services, irrespective of programming languages, operating systems, and network configurations. The interaction between two services can be accomplished using an activity known as Invoke, which is a BPEL component used to specify the operations of the service to be executed. Such operations are identified using Partner links. This requires that the WSDL file for the target service be assigned to the Partner Link property for the Invoke activity. In such an interaction, there is a possibility that the target service may become unavailable due to technical issues, including a failure in the system, updating procedures, or high load of executions, which can cause the process to crash and throw exceptions. When an execution fails due to lack of availability, the reliability of the enterprise system can be affected, potentially violating Service Level Agreements, thus resulting in financial penalties or customer dissatisfaction. Therefore, it has become imperative to develop architectures that determine the most suitable service and avoid any failure, thereby enhancing the system's performance. This has become particularly important with the emergence of Cloud computing, which, in order to deliver reliable systems, requires sophisticated management procedures to ensure performance, robustness, depend ability and security [16].

In [4], a model-driven approach was introduced as a solution to the coordination of the invocations of Web services using a new service, known as the Protocol Service. It is designed as the front end of SoA services, and it is responsible for handling all invocation requests. The approach is based on the automatic runtime

replacement of the WSDL file when the target service has become unavailable or has been overused.

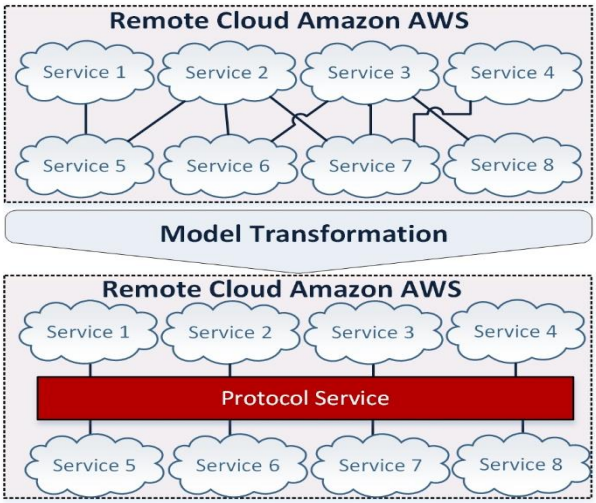

Fig. 5. The proposed Architecture with the Protocol service

Figure 5 depicts the outline of the model-driven approach. In this architecture, the integrated Protocol service coordinates interactions between services. The method requires that all requests be carried out using the Protocol service, with each service source providing the name of the target service to the Protocol service. The Protocol service then checks all services matching the request. If many services offer the same role, the Protocol service executes the one identified as being the most effective in terms of performance and availability.

In [3], an extension was introduced to consider access control policies. Such policies are necessary for building interactive enterprise projects based on the SoA framework (with the approach based on the enhancement of the Protocol service) in order to consider access control policies before carrying out the invocation. The Protocol service, prior to carrying out the invocation, checks whether the invoker is permitted to execute the required service. Therefore, a service known as an Access Control policies service (AC_Policies service) is introduced, which maintains the information related to access control policies. In this context, the Protocol service should interact with the AC_Policies service in order to determine the probability of accomplishing the invocation.

With the rapid emergence of Cloud computing, it has become essential to enhance the architecture dealing with the services deployed in the Cloud. Therefore, one of the aims of this paper is to extend the approach in order to apply it to the Cloud infrastructure, involving the automatic creation of a monitoring service that permits the user to supervise the assignment of the access control policies and the execution of Web services.

4. A Model-Driven Approach

In this paper, an approach to automating the integration of access control policies for Web services deployed over the Cloud. The approach also aims to provide a dynamic monitoring platform permitting users to supervise the execution of Web services and the assignment of the access control policies.

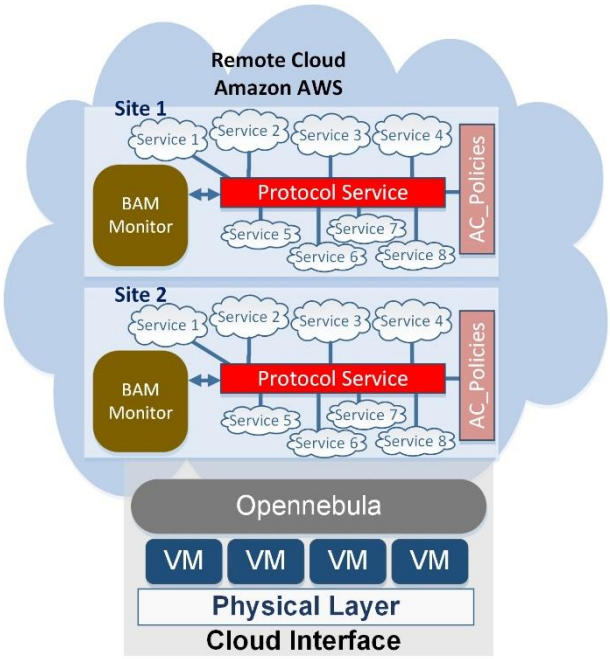

Fig. 6. An overview of the approach

Figure 6 shows the outline of the approach; a service referred to as an Access Control policies service (AC_Policies service) is introduced, which maintains information related to access control policies. This service is integrated with the Protocol service coordinating services deployed in the Cloud (e.g., on Amazon AWS). The Protocol service is designed to receive invocation requests from the source service, which are then forwarded to the target service. Each invocation request includes the name and the input values for the target service. The request is then validated by the Protocol service in order to determine whether the name of the service is valid, and guarantee provision of all the values to fit the required parameters of the destination

service. Based on the type of invocation, two options are then available in order to process the request: The first option is an asynchronous invocation (i.e. no result is expected from the target service) in which the Protocol service executes the target service and ends the process. In the second option, if the request involves two-way operations (synchronous), the Protocol service executes the target service and returns the results to the consumer.

In this context, the Protocol service is designed to handle all invocations passing through the Cloud interface, i.e., OpenNebula is used as an interface for the Cloud. OpenNebula is automatically configured to pass all the requests to the Protocol service, which then interacts with the AC_Policies service to determine the probability of accomplishing the invocation.

In addition to the access control policies, it is essential to monitor the execution of Web services and the assignment of the access control policies. In order to do so, the approach is extended to consider the automatic creation of a monitoring service, based on harnessing the capability of the MDA to automate the generation of the monitoring platform using the Oracle BAM. This framework can be used to build a real-time dashboard for monitoring the services of SoA. In particular, the approach extends the method of the generating of the Protocol service presented in [3] to include a BAM service.

4.1. *The Automatic Generation of the Protocol Service*

A model-driven method is used to automate the creation and integration of the Protocol service. The outline of this approach is depicted in Figure 7. The method requires passing all WSDL files of services and their XSDs as inputs. The set of Invoke activities are extracted from each file, after which the Partner link for each Invoke activity is automatically replaced with the Partner Link for the Protocol service. For example, Figure 8 depicts a constructor of an Invoke activity used to execute a service known as *CustomerService*: it is automatically modified by assigning the WSDL file of the Protocol service to the Partner Link property of the Invoke activity, as depicted in Figure 9.

In order to complete the task, the name of the target service needs to be assigned, along with its inputs to the Protocol service. Thus, the Assign activity precedes the Invoke activity, and it is used to assign the inputs required by the target service, while also being modified in order to assign the inputs and the name of the target service to the Protocol service. Figure 10 provides an example of an Assign activity used to copy the variable known as *CustomerID* to the input variable of an Invoke activity named *FindCustomerInfo*.

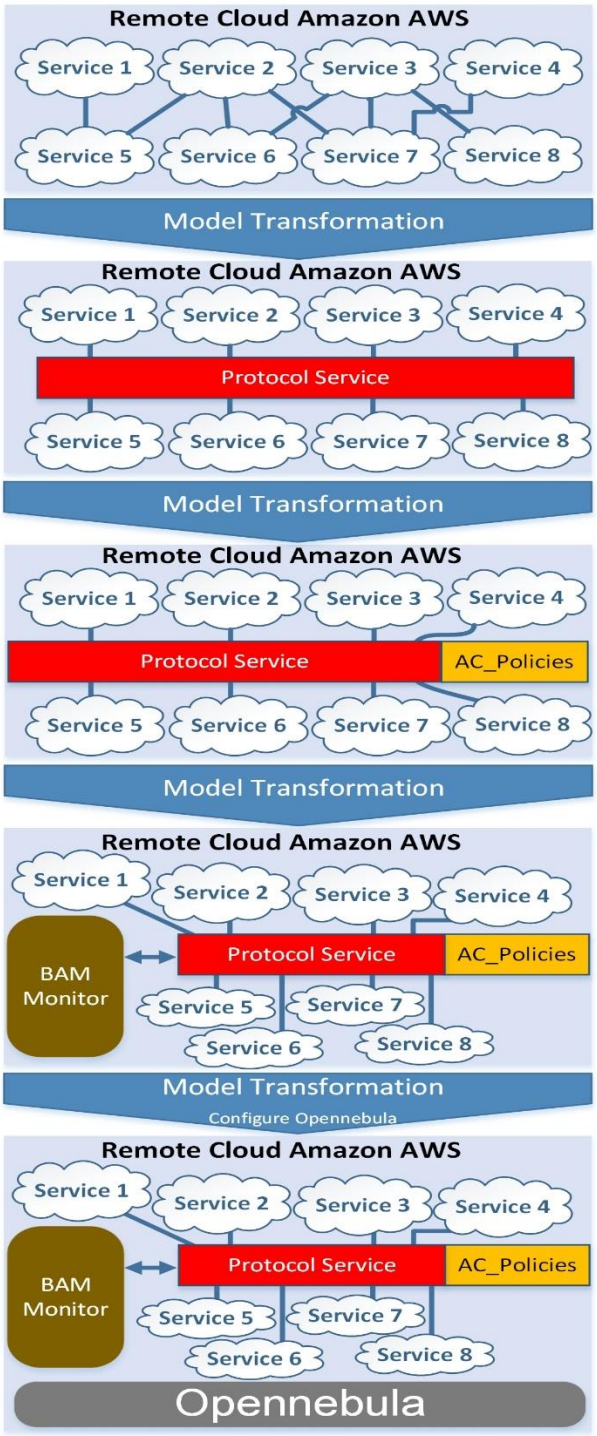

Fig. 7. Outline of the Model Transformation method

```
<invoke name="CheckCustomerAccount"
partnerLink="CustomerService"
portType="ns1:CustomerService"
operation="CheckCustomerAccount"/>
```

Fig. 8. Constructor of an Invoke Activity

The automatic modification of the Assign activity is facilitated with a model transformation requiring the BPEL metamodel, (as depicted in Figure 12). Figure 11 provides the QVT transformation rule to modify an Assign activity.

```
<invoke name="CheckCustomerAccount"
partnerLink="ProtocolService"
portType="ns1:ProtocolService"
operation="CheckCustomerAccount"/>
```

Fig. 9. Replaced Constructor of the Invoke Activity depicted in Figure 8

For more information regarding the automatic generation of the Protocol service, refer to [4].

```
<assign name="AssignID">
   <copy>
     <from variable="CustomerID"/>
     <to variable="FindCustomerInfoInput"/>
   </copy>
</assign>
```

Fig. 10. Constructor of an Assign Activity

4.2. *Automatic Integration of Access Control Policies*

The approach presented in this paper aims to extend the previous approach [3] to consider the automatic process of assigning access control policies to Web services. It also involves the automatic generation of a monitoring service that can be used to supervise the assignment of the access control policies and the execution of Web services. The key concept behind this extension is based on the generation of an AC_Policies service able to act as a platform for assigning access policies. Then, AC_Policies service is linked with the Protocol service. These services are eventually linked with a monitoring service that enables the user to monitor the process of the assignment of access control policies.

```
mapping Assign::assign2assign() : Assign
{   name := self.name;
    foreach(e Element | copy:Copy)
    {
      e.form.variable=e.form.variable;
      e.to.variable="ProtocolServiceInput";
    }}
```

Fig. 11. Transformation Rule for Modifying Assign Activity

The outline for using the Protocol service and the AC_Policies services is as follows: 1) The Protocol service receives an invocation request from the source service; 2) It obtains the invoker details along with the destination details; 3) It sends a query request to the AC_Policies service in order to determine whether this invocation is permitted before proceeding further; 4) If permitted, the Protocol service forwards the request to the target service; 5)Each invocation request then involves the name of the target service and the input values for the target services; 6) This request is subsequently validated by the Protocol service in order to determine whether the name of the service is valid and to ensure the provision of all the values for the required parameters of the destination service; 7) Based on the type of invocation, there are two options for processing the received request. If it is an asynchronous invocation (i.e., no result is expected from the target service), the Protocol service executes the target service and terminates the process. If the request involves two-way operations (synchronous), the Protocol service executes the target service and the result is returned to the consumer.

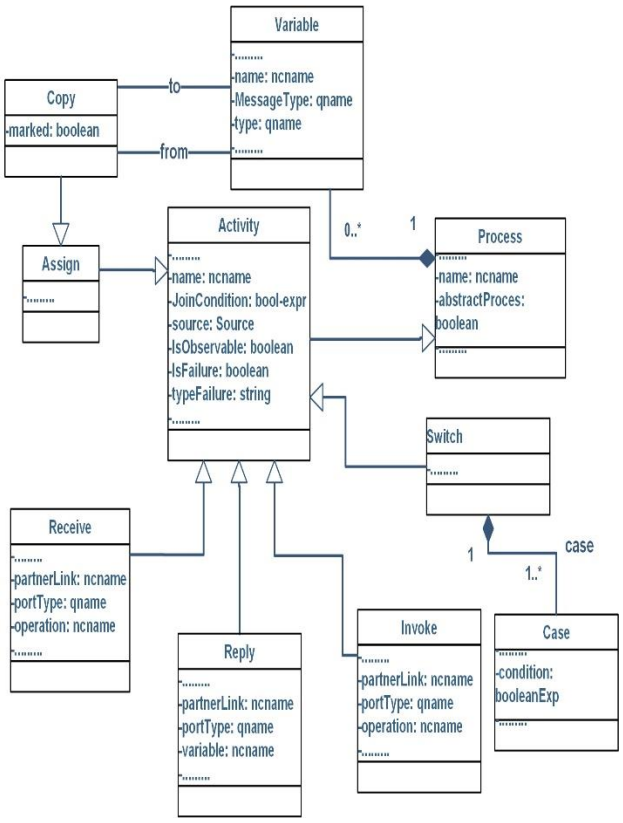

Fig. 12. The BPEL metamodel

4.2.1. Web Service Access Control Policies

The proposed method enables the user to assign access control policies via an automatically generated interface permitting the assignment of access control policies for Web services deployed over the Cloud. The access control policy's form of this approach is divided into three parts: 1) Requester, 2) Provider, and 3) Policy (Figure 13). The Requester can be assigned to either a system user or a service, but the Provider needs to be assigned to a service. The Policy involves two letters: 1) The first indicates permission for the execution of the Provider service by the user/service assigned to the Requester part. If it is assigned to "x", it is allowed to carry this invocation, and if it is assigned to "-", it is not permitted.2) The second indicates whether this invocation is an asynchronous operation (i.e., no result is expected from the target service), or a two-way operations (synchronous) (i.e., a result should be returned to the consumer represented by the letter "u").

$$\underbrace{user/service}_{\text{Requester}} :: \underbrace{service}_{\text{Provider}} :: \underbrace{xu}_{\text{Policies}}$$

Fig. 13. Access Control Policy Structure

For example, assuming the existence of a service called "*service1*" with two users (named "*user1*"and "*user2*") belonging to a group of users known as "*group1*", the potential policies are as follows:"*service1*" is permitted to access "*service2*", with the execution returned to the invoker. This policy is represented as follows (note that "*service1*" can be replaced by a user (such as "*user1*"), which is explained in a similar manner):

```
service1::service2::xu
```

Where "*service1*" is permitted to execute "*service2*" (but without being permitted to return any result to the invoker), the access control policy is presented as follows:

```
service1::service2::x-
```

Therefore, if "*service1*" is not permitted to execute "*service2*", the access control policy is depicted as follows:

```
service1::service2::--
```

Such policies can also be attached to groups. The following example demonstrates the ways in which the users of a group named "*group1*" are permitted to execute and receive results from "*service1*".

```
group1::service1::xu
```

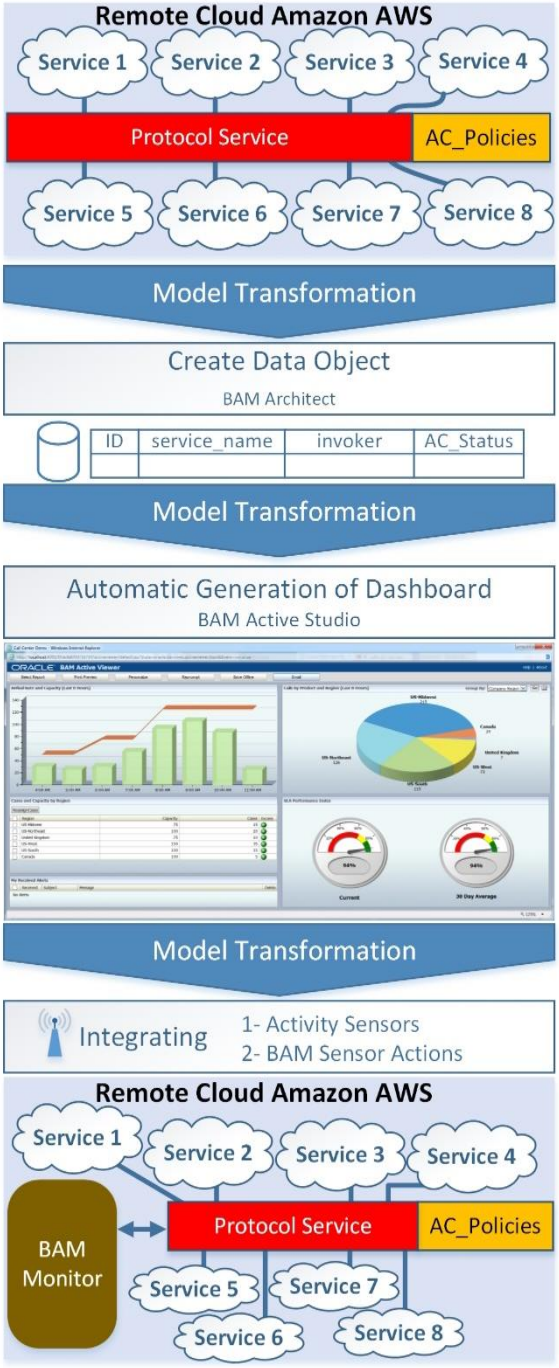

Fig. 14. Outline of the automatic generation of the monitoring service

4.3. *Generation of the Monitoring Service*

It subsequently becomes essential to monitor the process of assigning the access control policies, along with the execution of Web services. The monitoring service is designed to visualize the assignment of policies and the execution of Web services, and to ensure the delivery of reliable and dependable enterprise applications with a higher quality and within budget. This approach harnesses the capability of MDA to automate the generation of the service.

The service produced by this approach is based on Oracle BAM, which provides a mechanism that can be used to build a real-time dashboard for monitoring services deployed within the SoA framework. In particular, it involves the creation and integration of a BAM service used to monitor interactions undertaken by the Protocol service.

Figure 14 outlines the process of generation and the integration of the BAM service. These steps are described in detail in the following sections.

4.3.1. The Automatic Creation of the Data Object

A Data Object is a BAM component that is used to maintain data received from SoA services. Each Data Object is comprised of a table designed to store information required for presentation in BAM dashboards. Columns can be specified and added to the Data Object, with each column having a specific type. In this approach, the data object is created with four columns: 1) ID, 2) service_name, 3) invoker, 4) AC_Status. The column named "service name" is used to capture the name of the target service, while the "invoker" field is used to capture the name of the source service, and the access control policies status is stored in the "AC_Status" field.

The Data Object is automatically generated and integrated into the BAM server (Figure 14). Automatic creation is accomplished using model transformation, which requires the meta-model of the relational database.

4.3.2. Automatic Integration of Sensors

Monitoring the assignment of the access control policies and the execution of the Web services requires collecting specific data from SoA services relating to the executions. Oracle BAM provides the user with a number of ways in which data can be collected efficiently from SoA services, such as BAM Adapter, Java Message Service (JMS), MQSeries, and BAM Sensor Action. In the current approach, BAM sensors are used to collect and send the required data from SoA services to the BAM dashboard by linking a sensor to each Invoke activity included at the Protocol service.

Two main steps needs to be undertaken in order to link a sensor with an Invoke activity (Figure14). First, an Activity Sensor is automatically created and named using the title of the Protocol service file concatenated with "_sensor.xml". This file involves the declaration of activities that should be monitored, along with the identification of variables, such as data to be passed to the BAM server. More specifically, each activity sensor has a specific name assigned to the *sensorName* attribute. The *target* attribute is used to identify the activity linked to this sensor. The evaluation time of each sensor is also declared and assigned to *evalTime*. The value can be assigned as follows: 1) at the activation of the activity; 2) after completing the execution of the activity; 3) when a fault is thrown during the execution; or 4) during a compensation or retry situation. For example, assigning "activation" to the *evalTime* property ensures that the sensor is triggered when the activity is initiated, while assigning "all" ensures that the sensor is triggered if any of the cases have occurred. The configuration of the sensor also requires identification of the variables whose data will be passed to the BAM server to be stored in the Data Object and displayed in the dashboard. Figure 15 depicts a simple example: the declaration of a sensor named *ActivitySensorAssign* linked to an Invoke activity named Invoke *GetCustomerInfo*. The evaluation time of the sensor is assigned to "all", and the data to be sent to the BAM server after firing the sensor is assigned to the input variable.

```
<sensor sensorName="ActivitySensorAssign"
  classname="BpelActivitySensorAgent"
  kind="activity"
  target="Invoke_GetCustomerInfo">
     <activityConfig evalTime="all">
     <variable outputDataType="process"
     target=
"\$inputVariable/payload/client:process"/>
     </activityConfig>
</sensor>
```

Fig. 15. A simple example of an activity sensor

The second step in the creation of sensors is to link each Activity sensor with a BAM Sensor Action. This process is used to send data to the BAM server by mapping variables in a business process to their

equivalent fields in a Data Object. The creation and configuration of a BAM Sensor Action is accomplished as follows: 1) The *Action Name* of both the BAM Sensor Action and the Sensor value are specified. The *Action Name* can be assigned to a significant title, while the Sensor value is assigned to the name of the Activity Sensor created in the previous step. The sensor action requires assigning a Data Object that is linked to the one has already been created in Oracle BAM. 2) The operation to insert information into the Data Object needs to be identified. It can be: *Upsert* (a merge operation); *Insert* (used in this approach); *Update*; or *Delete*. Moreover, the BAM Connection Factory Java Naming and Directory Interface (JNDI) value (which identifies the runtime server connection pool) is specified. In this context, the JNDI value is assigned to "*eis/bam/soap*". 3) The Map File, used to map BPEL data values to their corresponding fields in the BAM Data Object, is created using the XSL transformation mapper.

4.3.3. Generation of the BAM Dashboard

The BAM Active tool is used to create the dashboard report (see Figure 16 for a snapshot of the BAM dashboard). In this approach, the dashboard is designed to present a number of charts showing important information related to access to the system services. For example, the dashboard reveals the number of denied executions for each service, along with a number of percentages related to the access policies violations.

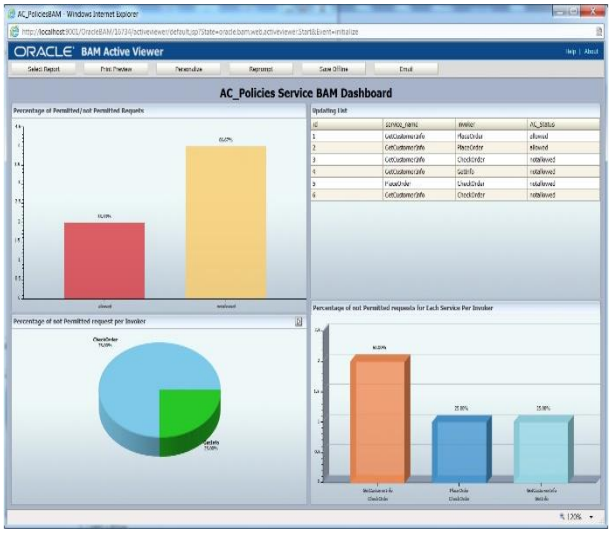

Fig. 16. Real-Time BAM dashboard report

5. Implementation of the Current Approach

The approach introduced in this paper is implemented as an Oracle JDeveloper plugin. The implementation follows the method outlined and depicted in Figures 7 and 14. This method requires passing all BPEL files and their XSDs as inputs. All Invoke activities are extracted for each BPEL service. The Partner link for each Invoke activity is then automatically replaced with the Partner Link for the Protocol service. The AC_Policies service is automatically generated and linked with the Protocol service.

The administrator interface is implemented as a Java Graphical User Interface (GUI), which permits the user to specify different policies for system services based on groups or individuals. Figure 17 shows a snapshot of the tool, divided into two parts. The first part is used to specify policies based on users, where a user and a target service are selected from the combo box of the users and services. Subsequently, the tool fetches the current values and waits for the update from the administrator. The second part involves on assigning policies based on groups.

Finally, the BAM dashboard is automatically generated and configured in the system. The BAM dashboard is linked to the Protocol service, which updates the dashboard after each execution.

Fig. 17. AC_Policies assignment interface

6. Discussion and related works

Access control policies have been studied generally in the literature, and most of the proposed solutions are based on logic expressions. A method known as WS-Policy4MASC [19] was introduced as an extension of Web Service Policy (WS-Policy) [11] to enable the Manageable and Adaptive Service Compositions (MASC) system. The purpose of the method is to discover either functional exceptions or QoS deviations by analyzing how a system behaves with respect to the interaction of client side and provider side policies, as expressed using WS-Policy. Ardagna et al. [10] proposed an approach based on using (WS-Policy) framework that involves describing and communicating the policies of a Web service to represent its approach.

In [39], a framework known as Authorization-Based Access Control (ABAC) was proposed. This system provides locally verifiable authorizations and delegation tracking that are compatible with common Web tools. This technique, which is considered reusable, is distributed and meets the scaling requirements of large distribution services.

Extensible Access Control Markup Language (XACML) [1], proposed as an access control language for an open-world scenario, provides flexibility and interoperability. This approach is an XML-based language for expressing and interchanging access control policies. Ardagna et al. [9] have provided a simple and effective framework for enforcing the access control paradigm in order to provide an expressive solution that can be deployed in the XACML standard.

7. Conclusion

The approach described in this paper introduces a method for monitoring the assignment of access control policies. The approach harnesses the capability of MDA to automate the generation of a BAM service integrated with a Protocol service generated to coordinate invocations between services. The Protocol service described is linked to an Access Control policies (AC_Policies) service, which is produced automatically to facilitate the assignment of access control policies to Web services. The Protocol Service, AC_Policies service, and BAM service are automatically produced by parsing the original web services; thus, they generate a set of modified services with an integrated Protocol service, an AC_Policies service, and a BAM service. This approach was implemented as a Plugin.

References

1. eXtensible Access Control Markup Language (XACML) Version 2.0. Technical report, OASIS Access Control TC, February 2005.
2. D. H. Akehurst, B. Bordbar, M. J. Evans, W. G. J. Howells, and K. D. McDonald-Maier. Sitra: Simple transformations in java. *In Models'09*, volume 4199, pages 351–364, Italy, 2006.
3. Mohammed Alodib. An approach to automating the integration of the access control policies for web services. In *14th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD2013),* USA, 2013.
4. Mohammed Alodib. A framework for the coordination of the invocations of web services. *In The First International Conference on Building and Exploring Web Based Environments*, Spain, 2013.
5. Mohammed Alodib, Behzad Bordbar, and Basim Majeed. A model driven approach to the design and implementing of fault tolerant service oriented architectures. *In IEEE International Conference on Digital Information Management (ICDIM),* pages 464–469,London, 2008.
6. Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services Concepts, Architectures and Applications*. Springer, 2004.
7. Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. UML2Alloy: A Challenging Model Transformation. *In ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2007)*, 2007.
8. Arcstyler. Arcstyler 5.0- interactive objects. 2005.
9. Claudio A. Ardagna, Sabrina De Capitani di Vimercati, Stefano Paraboschi, Eros Pedrini, Pierangela Samarati, and Mario Verdicchio. Expressive and deployable access control in open web service applications. *IEEE Trans. Serv. Comput.*, 4(2):96–109, 2011.
10. Claudio Agostino Ardagna, Ernesto Damiani, SabrinaDe Capitani di Vimercati, and Pierangela Samarati. A web service architecture for enforcing access control policies. Electron. *Notes Theor. Comput. Sci.*, 142:47–62, 2006.
11. Siddharth Bajaj, Don Box, Dave Chappell, Francisco Curbera, Glen Daniels, Phillip Hallam-Baker, Maryann Hondo, Chris Kaler, Dave Langworthy, Anthony Nadalin, Nataraj Nagaratnam, Hemma Prafullchandra, Claus von Riegen, Daniel Roth, Jeffrey Schlimmer, Chris Sharp, John Shewchuk, Asir Vedamuthu, U¨mit Yalcinalp, and David Orchard. Web services policy 1.2- framework (WS-policy). Technical report, W3C, 2006.
12. Christian Baun, Marcel Kunze, Jens Nimis, and Stefan Tai. *Cloud Computing: Web-Based Dynamic IT Services*. Springer Publishing Company, Incorporated, 1st edition, 2011.

13. David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web services architecture. *W3C*, 2004.
14. Behzad Bordbar, Gareth Howells, Michael Evans, and Athanasios Staikopoulos. Model transformation from owl-s to bpel via sitra. *In Proceedings of the 3rdEuropean conference on Model driven architecture foundations and applications, ECMDA-FA'07*, pages43–58, Berlin, Heidelberg, 2007. Springer-Verlag.
15. Project Caroline. http://research.sun.com/projects/caroline [retrieved: April, 2015].
16. S.A. De Chaves, R.B. Uriarte, and C.B. Westphall. Toward an architecture for monitoring private clouds. *Communications Magazine*, *IEEE*, 49(12):130–137,2011.
17. Jean Dollimore, Tim Kindberg, and George Coulouris. *Distributed Systems: Concepts and Design (4th Edition) (International Computer Science Series)*. Addison Wesley, 2005.
18. Troy Bryan Downing. *Java RMI: Remote Method Invocation*. IDG Books Worldwide, Inc., Foster City,CA, USA, 1st edition, 1998.
19. Abdelkarim Erradi, Piyush Maheshwari, and Vladimir Tosic. Ws-policy based monitoring of composite web services. *In Proceedings of the Fifth European Conference on Web Services*, pages 99–108, Washington,DC, USA, 2007. IEEE Computer Society.
20. Django Web Framework. http://www.djangoproject.com, [retrieved: April, 2014]
21. David S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, 2003.
22. B. Furht and A. Escalante. *Handbook of cloud computing*. Computer science. Springer US, 2010.
23. Richard Grimes and Dr Richard Grimes. *Professional Dcom Programming*. Wrox Press Ltd., Birmingham, UK, UK, 1997.
24. Matjaz B. Juric, Benny Mathew, and Poornachandra Sarang. *Business Process Execution Language for Web Services*. Packt Publishing, 2004.
25. kermeta. http://www.kermeta.org/,[retrieved:April, 2014].
26. A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture Practice and Promise*, Addison-Wesley, 2003.
27. H. Kreger. Web services conceptual architecture. IBM Software Group, 2001.
28. Ruben Lara, Holger Lausen, Sinuhe Arroyo, Josde Bruijn, and Dieter Fensel. Semantic web services: description requirements and current technologies. *In ICEC*, 2003.
29. Frank Leymann. Web services: Distributed applications without limits. *In 10th Conference on Database Systems for Business, Technology and Web (BTW'03)*, pages 26–28, Leipzig, 2003.
30. MOF. Meta object facility (mof) 2.0 core specification, object management group, 2004
31. OBEO, INRIA. Atlas transformation language. http://www.eclipse.org/atl/, [retrieved: April, 2015].
32. OMG. http://www.omg.org/corba/.
33. OMG. MOF QVT Final Adopted Specification, 2005.OMG doc.
34. OMG. OCL 2.0, 2006.
35. Michael P. Papazoglou. A survey of web service technologies, 2004.
36. H. Petritsch. Service-oriented architecture (soa) vs. component based architecture. Technical report, Vienna University of Technology, Vienna, 2006.
37. A. Reynolds and M. Wright. *Oracle SoA Suite 11g R1 Developer's Guide*. 2010.
38. Roger Sessions. *COM and DCOM: Microsoft's vision for distributed objects*. John Wiley & Sons, Inc., New York, NY, USA, 1998.
39. Gayatri Swamynathan, Tyler Close, Sujata Banerjee, and Rick McGeer. Scalable access control for web services. *In the IEEE International Conference on Creating, Connecting and Collaborating through Computing*, pages 93–98, 2007.
40. Henry S. Thompson, David Beech, Murray Maloney,and Noah Mendelsohn. Xml schema part 1: Structures,2004.
41. P. Wang. *Oracle BAM 11g R1 Handbook*. Packt Publishing,Limited, 2012.
42. Steve K. Wood, David H. Akehurst, Oleg Uzenkov, W. G. J. Howells, and Klaus D. McDonald-Maier. A model-driven development approach to mapping uml state diagrams to synthesizable vhdl. *IEEE Trans.Comput.*, 57:1357–1371, 2008.