A Framework for the Coordination of the Invocations of Web Services

Mohammed Alodib
Qassim University
P.O BOX 385
Buraidah 51411
Qassim, Saudi Arabia
alodib@qu.edu.sa

***Abstract*—Coordinating the various Web services invocations is one of the key challenges of Service oriented Architectures. When services fail due to lack of availability this may violate Service Level Agreements causing financial penalties or customer dissatisfaction to providers. Therefore, it is crucial to develop a method of on-line coordination of these invocations in order to enhance the performance of the systems in place and avoid the overuse of services. This paper aims to present a Model Driven Architecture (MDA) approach to the automated creation and integration of Protocol Services, which are deployed with the system to coordinate invocations between services. The outline of this method is as follows. Business Process Execution Language (BPEL) models of services are parsed and the PartnerLink for each Invoke activity is assigned to the Web Services Description Language (WSDL) file of the Protocol Service using MDA transformations. Then, the Protocol service is computed, generated and integrated automatically into the system. As a proof of concept, an implementation of the suggested approach was created, in the form of an Oracle JDeveloper plugin that automatically produces new Protocol services and integrates them with existing services.**

Keywords-Web services; Quality of Service; Coordinating; Model Driven Architecture

I. INTRODUCTION

Service oriented Architecture (SoA) is a framework which provides a layered architecture for organising software resources as services, so that they can be deployed, discovered and combined to produce new services [1]. In real-world business processes, it is crucial to develop architectures to discover the most suitable service in order to avoid excessive use of services, and so enhance the performance of the system.

In the current version of SoA, an invocation request is processed using BPEL activity known as Invoke. This requires assigning a WSDL file of the target service to the Partner Link property of the Invoke activity. If the destination service becomes unavailable for any reasons, this may cause distraction to other services and lead to customer dissatisfaction. In addition, a service may become slow in its responses due to the uncoordinated overuse of its operations by other services. To resolve such issues, the WSDL file for the failed service, or the slow service associated with the Invoke activity can be manually replaced with another WSDL file, one designed for a service that gives the same result, but it is deployed by a different provider. Therefore, a model-driven approach to automating this replacement is proposed.

The presented approach provides a dynamic technique to discover the best available service using a simple genetic algorithm. This algorithm is based on ranking Web services using the previous invocations history. In this approach, all the invocations are forwarded to a *Protocol service*, which works as a coordinator controlling all invocations. The the Protocol service is initiated by the request from the consumer; it then forwards the request to the target service, obtaining the result from the provider, and returning the result to the consumer.

From a performance perspective, this architecture can potentially result in a bottleneck, as all invocations should be processed by the Protocol service. However, the Protocol service is distributively generated and integrated into the system; i.e. each site has its own Protocol service, which controls the internal invocation requests. When faced with an external invocation for a remote service, i.e. deployed at different server, the Protocol service interacts with the Protocol service located at the external site by forwarding the invocations.

This paper is organised as follows. Section III-A presents a brief review of Service oriented Architecture (SoA). Section III-B reviews the Web Services. An introduction to the fundamentals of the Business Process Execution Language (BPEL) is described in Section III-C. Section IV presents the principles of Model Driven Architecture (MDA). The description of the problem is discussed in Section V. Section VI presents the solution, which is implemented as an Oracle JDeveloper's plugin.

II. DISCUSSION AND RELATED WORKS

Yan et al. [2], [3] proposed a method to monitor Web services in order to trace faults and recover from their effects. Their method utilises Model-Based Diagnosis (MBD) theory [4], which provides techniques to monitor static and dynamic systems using partial observations. Such methods require in-depth knowledge of the system. Their method is designed to monitor failures, such as mismatching parameters when occurrences are thrown up as exceptions. On the other hand,

 ISBN: 978-1-61208-248-6

our approach aims to deal with monitoring and coordinating the invocations in order to avoid failed or overused services. Our goal is to maintain the system at a high level of performance by avoiding dynamically failed or overused services.

Ardissono et al. [5] also proposes a model-based approach to monitor and diagnose Web services. Their approach aims to provide self-healing services, which guarantee autonomous diagnostic and recovery capability. This approach is based on adopting grey-box models for each Web service to expose the dependency relationships between the input and output parameters to the public. The dependency relationships are used by Diagnosers to determine the service which results in exceptions.

In [6], [7], we present approaches to dealing with monitoring failures caused by undesirable scenarios, such as Right-First Time (RFT) Failure, which occurs when a business process fails to complete a task the First-Time and is forced to repeat a part of the task again (i.e., when a task is executed more than once, indicating incorrect execution of the task in the first place, or the invocation of an erroneous execution). Such occurrences of failure may result in violations of Service Level Agreements (SLA).

III. PRELIMINARIES

A. Service oriented Architecture (SoA)

SoA is directed towards the implementation of business processes via the composition of interactive services [1]. In general, SoA is a prevailing software engineering product, which ends the domination of traditional, distributed system platforms [8]. The growth rate for SoA use in industry has been estimated at over 24%, as measured between 2006 and 2011 [9], and the rapid movement towards SoA has been encouraged by the positive results already recorded; for example, the level of reusability in SoA has, on average, been enhanced to more than 2.5 times that of non-SoA development.

A simple SoA infrastructure involves three independent collaborative components, which are described below [10], [11]; see Figure 1:

- ***Service provider***: The service provider is responsible for publishing the services, and is the owner of the services; e.g. companies and organisations.
- ***Service requester***: A requester is a client or organisation that wishes to make use of a service that is being provided. The requester searches for the Web services desired from the service registry.
- ***Service registry***: A global registry acts as a central service which provides a directory where service descriptions are published by the Service Provider. Then, *Service Requesters* find service descriptions in the registry and obtain binding information for services from the *Service Provider*.

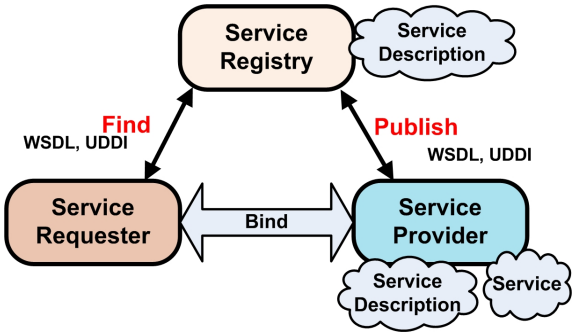

Figure 1. A Basic Service Oriented Architecture [11]

B. Web Services

Web services offer a preferred solution to the problem of integration among autonomous and heterogeneous software systems [12]. They are well-defined, self-contained, loosely coupled, self-describing, modular applications that can be published, located and invoked across the Web network [12]. These features mean that Web services have the ability to be dynamically invoked by other applications or other Web services, and are composed in tandem with other services to achieve complex tasks. In other words, Web services are highly reusable components, acting as building blocks to develop service composition, as well as to solve the application communication and integration issues.

The development of a composite web service is built upon the Service oriented Architectural paradigm [13]. Communication in a composition of Web services is based on the use of well-accepted standards and the XML messaging framework [14]. Such standards can be used to encapsulate the service's business logic and functionality in order to expose the functionality only, not the implementations via the accessible interfaces. Therefore, application programs communicate with one another irrespective of their programming language, operating system and hardware platforms.

Web services communicate using common Extensible Markup Language (XML), XML Schema Definition XSD [15] and standard TCP/IP based communication protocols. Moreover, various XML-based standards are used by Web services in order to describe their architecture, intercommunication, collaboration and discovery [12]. In particular, the communication messages between a Service Requester and a Service Provider are encoded into *Simple Object Access Protocol (SOAP)* messages, which are plain text XML messages. The *Web Services Description Language (WSDL)* is used to describe the invocation details of a Web service, such as the service name, the operations available, and the information related to the input and output variables. The *Universal Description Discovery and Integration (UDDI)* provides protocols for querying and updating Web service information. For communication purposes, Web services

 ISBN: 978-1-61208-248-6

utilise existing standard TCP/IP protocols such as HTTP, HTTPS, SMTP, and FTP [16].

C. Business Process Execution Language (BPEL)

In the past few years, SoA has been adopted and widely used by the IT industry. One of the most popular standards of such adaptations is Business Process Execution Language for web services (BPEL) [17]. BPEL is a modelling language used to specify a sequence of actions that take place within business processes in order to generate enterprise applications. BPEL offers a rich number of diagrammatic notations ideal for supporting the modelling of complex behaviours; i.e. sequential, parallel, iterative and conditional. In addition, similar to traditional programming languages, BPEL offers constructs, in the form of loops, branches, variables and assignments.

Business Process Execution Language for Web Services (BPEL, WS-BPEL, BPEL4WS) is a graphical language that is used for the composition, orchestration, and coordination of Web services [1]. Combining and linking existing Web services and other components to deliver new composition services is referred to as *business processes*; therefore, BPEL is used to specify a set of actions within business processes, in order to achieve a common business goal. The BPEL specification is based on the Web Services Description Language (WSDL) [18], which is an XML language describing services as a set of accessible interfaces, for producing business processes that support interoperability [19].

IV. Model Driven Architecture (MDA)

Model Driven Architecture (MDA) [20], [21] is a framework introduced by the Object Management Group (OMG) in order to promote the role of modelling in software development. One of the main goals of MDA is model transformation; a process whereby models in a source language are mapped so as to be captured in the destination language. In the MDA context, model transformation is defined by a number of transformation rules, which specify the mapping of the *meta-elements* of the constructs of the *metamodel* of the *source language* into the *meta-elements* of the *destination language*. The metamodels of the source and the target language are specified using a common language, called the Meta Object Facility (MOF) [22]. In general, models in the MDA are instances of metamodels.

Meta Object Facility (MOF) Query/View/Transformation Specification (or QVT for short) [23] 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*. *Relations language* is a high level language that provides a textual and graphical notation for the purpose of defining the mappings, while *Core language* is a small language based on Essential MOF (EMOF) and OCL, which is used to support pattern matching and the evaluation of conditions. QVT *Operational Mappings language* is a high level imperative language that extends Object Constraint Language (OCL) [24] with essential features (such as the ability to define loops) in order to write complex transformation rules [23]. In this study, we used QVT Operational Mapping language to obtain the specifications for the transformation rules.

The QVT *Operational Mapping* language is specified as a standard method for providing imperative implementations. This language is based on using MOF as a repository for metamodels. The general syntax for the body of an Operational Mapping is depicted in Figure 2, where the *source* is the source of the model transformation. The *mappingFunction* is the name of the model transformation, which may require some inputs, 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 completes. The *'when'* part has a Boolean expression that should be verified as true before commencing the execution. The *'where'* part includes the conditions that have 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::mappingFunction(parms):target
  when {...}
  where {...}
  {
        init{...}
        population{...}
        end{...}
  }
```

Figure 2. The general syntax for the body of a mapping operation.

There are many industrial and academic case tools supporting model transformations, such as Kermeta [25], Arcstyler [26], OptimalJ [27], ATLAS [28] and SiTra [29]. In this paper, we will use the Simple Transformer (SiTra) [29] transformation engine to execute the transformation rules. SiTra is a lightweight Model Transformation Framework, which intends to use Java for both writing Model Transformations and providing a minimal environment for transformation execution.

V. Description of the Problem

From a SoA point of view, an interaction between two services can be performed with the help of an Invoke activity, which is a BPEL component used to specify the operations of the service that we intend to execute. Such operations are identified using Partner link. To achieve this, the WSDL file for the target service is assigned to the Partner Link

 ISBN: 978-1-61208-248-6

property of the Invoke activity. However, the target service may then become unavailable due to technical issues; such as a failure in the system, updating procedures, or the high load of executions, and this may cause the process to crash and throw exceptions. Consequently, it is critical to identify failed services, so that suitable remedial actions can be taken.

The typical method for resolving those issues caused by unavailable services is to manually replace the WSDL file of the service, as linked to the Invoke activity with another service providing the same functionality, but deployed by a different server. For example, assume that there are two services called *find_flight* and *FlightSearch*. These services provide the same functionality, and it is supposed that there is an Invoke activity used to execute the *find_flight* service. If we assume that the *find_flight* service becomes unavailable for any reasons, it becomes necessary to perform a recovery action so as to solve that issue. This can be achieved by replacing the current WSDL file of the service with another one, such as *FlightSearch*. Although this solves the problem, it is both a costly and time consuming solution as it should be carried out manually by a developer. Therefore, a dynamic approach to enhance and automate the process of this replacement is presented.

The approach presented proposes a framework that provides on-line automated modifications. In other words, the approach aims to provide dynamic executions based on the automatic runtime replacement of the WSDL file, in case the target service becomes unavailable or where it is already overused.

VI. The Model-Driven Approach

The approach presented here proposes a service intended for monitoring and coordinating interactions between services. The service introduced is referred to as the *Protocol service* and aims to discover the best available service, depending on its performance and availability. This method requires that all invocations between services are carried out using a Protocol service, whereby each source service provides the name of the target service to the Protocol service. Then, the Protocol service checks all the services that match the request received. From a performance and availability point of view, it then evaluates these services in order to find the most suitable service.

The basic idea of the Protocol service is that it receives an invocation request from the source service and forwards this to the target service. Each invocation request involves the name of the target service, the inputs values for the target service. This request is then validated by the Protocol service to check whether the name of the service is valid, and to ensure that all the values for the required parameters of the destination service are provided. Based on the type of invocation, there are two options for processing the request received. Firstly, if it is an asynchronous invocation, i.e., no result is expected from the target service, then

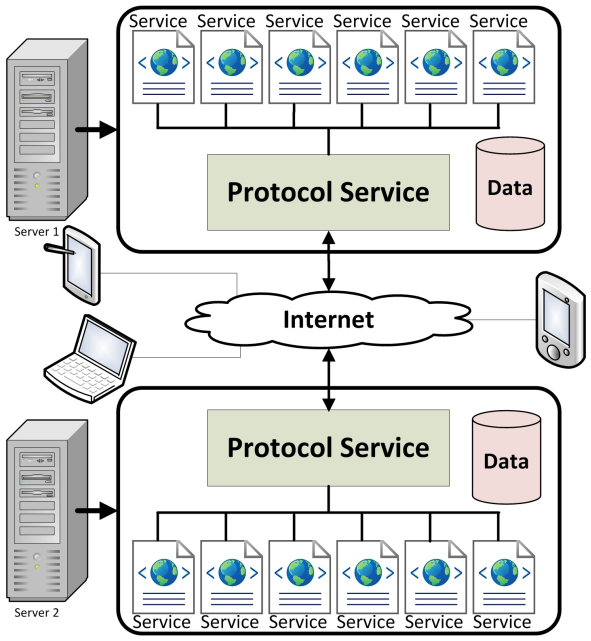

Figure 3. The proposed Architecture with the Protocol service

the Protocol service executes the target service and ends the process. Alternatively, if the request involves *two-way operations* (synchronous), the Protocol service executes the target service and the result is eventually returned to the consumer.

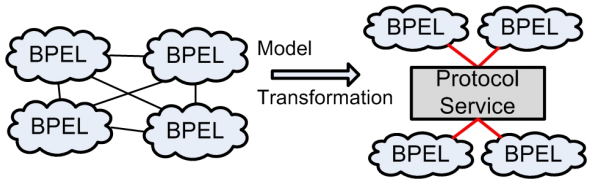

Figure 4. The outline of the Transformation method

The Protocol service is automatically and distributively generated for each site in the system as depicted in Figure 3. Each Protocol service is responsible for monitoring and coordinating interactions on the site in which it resides. An invocation between the two services deployed in two different sites requires that the Protocol service at the first site interacts with the Protocol service at the second site in order to complete the invocation. For example, suppose that we have two services (Service A and Service B) deployed at separate servers; Server 1 and Server 2 respectively. Assume that Service A intends to invoke Service B. To accomplish this invocation, Service A sends a request to the Protocol service in Server 1. This request includes details about the target Service. Next, the Protocol service at Server 1 forwards the request to the Protocol service at Server 2, which then carries out the invocation, and if the invocation is

 ISBN: 978-1-61208-248-6

a synchronous operation, it returns the result to the Protocol service at Server 1. Eventually, the Protocol service for Server 1 returns the results to Service A.

The evaluation task involves selecting the most suitable destination service as based on a simple genetic algorithm to rank Web services using the invocations history. This algorithm aims to check all services in order to identify the best services from a performance point of view. Due to the nature of SoA, which requires the assignation of a specific WSDL file of a service to each Invoke activity at the runtime, it is possible to note an excessive use of that service, despite the fact that there are other services offering the same functionality and that they can be used to avoid such an overuse. This may occur because there is no on-line coordinator for the distribution of the request received for the available services. Therefore, it can be seen that it is necessary to control routing requests between services in a balanced manner, and this is achieved using the Protocol service.

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

Figure 5. A Constructor of an Invoke Activity

VII. Integration of the Protocol Service

For already pre-existing projects, the approach presented can be integrated automatically using a model-driven technique, which is implemented as an Oracle JDeveloper plugin. The implementation follows the outline of the method as depicted in Figure 4. This method requires passing all BPEL files and their XML Schema Definition (XSD) as inputs. For each BPEL file, the set of Invoke activities are extracted. Then, the Partner link for each Invoke activity is automatically replaced with the Partner Link for the Protocol service. For example, Figure 5 depicts a constructor of an Invoke activity used to execute a service called *CustomerService*. This 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 6.

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

Figure 6. A replaced Constructor of the Invoke Activity of Figure 5

As discussed in Section VI, the Protocol service requires that the user assigns the name of the target service and its inputs in order to complete the process. For this reason the Assign activity precedes the Invoke activity, and is used to assign the inputs required by the target service, being also modified in order to assign the inputs and the name of the target service to the Protocol service.

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

Figure 7. A Constructor of Assign Activity

The Assign activity contains one or more *Copy* operations, which are used to copy data from one variable to another, as well as to construct and insert data using expressions [30]. Figure 7 presents a simple example of a construct for an *Assign* used to copy the value of *CustomerID* to *FindCustomerInfoInput*. To accomplish the required modifications, each *'to'* property of the Copy operations of the Assign activity is replaced and it is assigned to the Protocol Service input variable. The following code depicts a snippet of code that is then used to map each Assign activity to a new Assign activity, where the *'to'* property of the Copy operation is assigned to the input variable of the Protocol service. The following QVT transformation rule depicts the specification of our transformation explained above:

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

VIII. Case Study & Evaluation

The presented approach is tested with the help of a simple case study described by Guillou et al. [31]. This example is based on a typical on-line e-shopping system consisting of three main services: Shop, Supplier and Warehouse.

As depicted in Figure 8, the customer accesses the Shop Web site to search for items. Then, he adds his items to the Shopping Cart which is then passed to the Supplier service by the Shop Service. For each item in the list, the Supplier service sends a request to the Warehouse to check if the item is available. If the item is available the Warehouse service sends an acknowledgement to the Supplier to complete processing the order. Next, the Supplier Service send back the list of available items to the Shop service. Finally, the list is forwarded to the customer who then confirms his order.

Evaluating the resources required to implement the Protocol service is considered a requisite task. Therefore, the

 ISBN: 978-1-61208-248-6

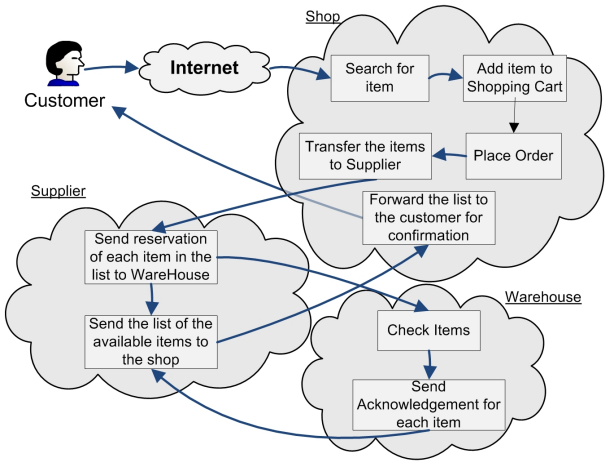

Figure 8. E-shopping Scenario

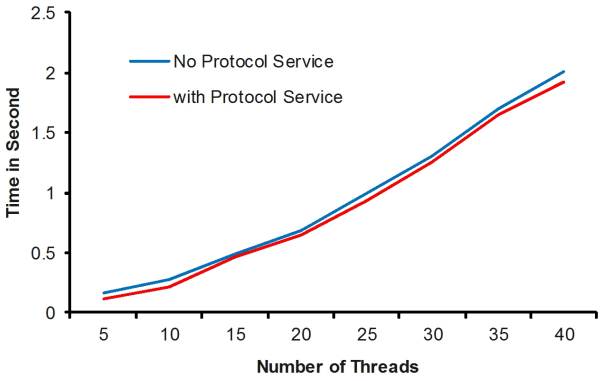

Figure 9. Stress Testing Result

approach presented here have been evaluated in terms of performance and it is compared with the traditional method which does not use the Protocol service. A common practise in evaluating services of SoA is to utilise the *Stress Test*. The *Stress Test* is a technique used to identify and verify the stability, capacity and the robustness of services [1]. The Stress Test requires defining the number of the concurrent threads that should be allocated to the service, the number of loops, and the delay between invocations. With the performance statistics, we can identify any possible bottlenecks and optimise performance.

The example is implemented in two different methods; one is based on the traditional way, i.e. without the Protocol service, and the second method is to use the presented approach, i.e., using the Protocol service. The Stress Test has been applied to these methods by handling a different number of concurrent threads. This is specified as 5, 10, 15, 20, 25, 30, 35, 40, 45 and 50 threads. The delay between invocations is assigned to one second. The machine which is used in this test has the following configuration: Lenovo W520, Intel Core i72820QM 2.30GHz processor, 16G RAM.

The mean of the executions time has subsequently been calculated and the results are depicted as a line chart in Figure 9. The performance of using the Protocol service can be seen as linear and parallel, and it shows better performance.

In addition to the performance, modularity can play a key role in the early design stages of the software architecture discipline [32]. Therefore, using the Protocol service provides a modularised design, which brings to the system the following benefits:

1) Reliability: using the *Protocol Service* provides faster and more reliable processes.
2) Faster and easier development. The focus would be on the functionality of code modules rather than on the mechanics of implementation.
3) Faster and easier testing.
4) Maintainability: this also makes modification of enterprise project easier.

Moreover, using the Protocol service supports the fact that this architecture increases the efficiency of the system as it is considered an Orchestration architecture which is a more flexible paradigm offering the following advantages over the Choreography [1]: i) the coordination of component processes is centrally managed by a known coordinator; ii) Web services can be incorporated without being aware that they are taking part in a business process; iii) alternative scenarios can be put in place in case of a fault. However, it could be argued that using the *Protocol Service* may result in bottlenecks affecting the performance of the system.

IX. Conclusion

This paper has presented a method of developing a service that can monitor the execution of invocations in a Service oriented Environment. The underlying concept relies on utilising the capability of MDA to generate a Protocol service, which coordinates invocations between services in order to enhance the performance of a system by avoiding failure or overuse of services. The automation of the creation of the Protocol service is based on parsing the original BPEL services, and generate a set of modified services with an integrated Protocol service. The approach presented is implemented as an Oracle JDeveloper plugin.

References

[1] M. B. Juric, B. Mathew, and P. Sarang, *Business Process Execution Language for Web Services*. Packt Publishing, 2004.

 ISBN: 978-1-61208-248-6

[2] Y. Yan, Y. Pencole, M.-O. Cordier, and A. Grastien, "Monitoring web service networks in a model-based approach," in *ECOWS05 (European Conference on Web Services)*, Sweden, 2005, pp. 192–203.

[3] Y. Yan and P. Dague, "Modeling and diagnosing orchestrated web service processes," in *IEEE International Conference on Web Services*, vol. 9, Salt Lake City, Utah, USA, 2007, pp. 51 – 59.

[4] W. Hamscher, L. Console, and J. de Kleer, Eds., *Readings in model-based diagnosis*. USA: Morgan Kaufmann Publishers Inc., 1992.

[5] L. Ardissono, L. Console, A. Goy, G. Petrone, C. Picardi, M. Segnan, and D. Dupre, "Cooperative model-based diagnosis of web services," in *In 16th International Workshop on Principles of Diagnosis*, Monterey, 2005, pp. 125–132.

[6] M. Alodib, B. Bordbar, and B. 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)*, London, 2008, pp. 464–469.

[7] M. Alodib and B. Bordbar, "A modelling approach to service oriented architecture for on-line diagnosis," *the journal of Service Oriented Computing and Applications*, pp. 1–17, 2012.

[8] D. W. McCoy and Y. V. Natis, "Service-oriented architecture: Mainstream straight ahead," Gartner Research, Tech. Rep., 2003.

[9] J. B. Hill, M. Cantara, E. Deitert, and M. Kerremans, "Magic quadrant for business process management suites," Gartner Research, Tech. Rep., 2007.

[10] M. P. Papazoglou, "A survey of web service technologies," 2004.

[11] H. Kreger, "Web services conceptual architecture." IBM Software Group, 2001.

[12] F. Leymann, "Web services: Distributed applications without limits," in *10th Conference on Database Systems for Business, Technology and Web (BTW'03)*, Leipzig, 2003, pp. 26–28.

[13] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services Concepts, Architectures and Applications*. Springer, 2004.

[14] "Semantic web services: description requirements and current technologies," in *In Proceedings of the International Workshop on Electronic Commerce, Agents, and Semantic Web Services held in conjunction with the Fifth International Conference on Electronic Commerce (ICEC)*, 2003.

[15] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, "Xml schema part 1: Structures," 2004.

[16] H. Petritsch, "Service-oriented architecture (soa) vs. component based architecture," Vienna University of Technology, Vienna, Tech. Rep., 2006.

[17] S. Blanvalet, *BPEL Cookbook: Best Practices for SOA-based integration and composite applications development*. PACKT PUBLISHING, 2006.

[18] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, "Web services description language (wsdl) version 2.0," 2006.

[19] BEA, IBM, Microsoft, A. SAP, and S. Systems, "Business process execution language for web services. version 1.1," 2003.

[20] D. S. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, 2003.

[21] A. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture- Practice and Promise,*. Addison-Wesley, 2003.

[22] MOF, "Meta object facility (mof) 2.0 core specification, object management group, available at www.omg.org,," 2004 [retrieved: 11, 2012].

[23] OMG, *MOF QVT Final Adopted Specification*, 2005, oMG doc. [retrieved: 11, 2012].

[24] ——, *OCL 2.0*, 2006, oMG doc. ptc/06-05-01 [retrieved: 11, 2012].

[25] kermeta, "http://www.kermeta.org/, [retrieved: 11, 2012]."

[26] Arcstyler, "Arcstyler 5.0- interactive objects." www.interactive-objects.com, 2005, [retrieved: 11, 2012].

[27] OptimalJ, "Compuware software coporation," 2005.

[28] OBEO, INRIA, "Atlas transformation language." http://www.eclipse.org/atl/, [retrieved: 11, 2012], 2005.

[29] D. H. Akehurst, B. Bordbar, M. J. Evans, W. G. J. Howells, and K. D. McDonald-Maier, "Sitra: Simple transformations in java," in *the 9th international conference on Model Driven Engineering Languages*, ser. LNCS, vol. 4199, Italy, 2006, pp. 351–364.

[30] IBM, Microsoft, *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*, OASIS, 2007.

[31] X. Le Guillou, M.-O. Cordier, S. Robin, and L. Rozé, "Chronicles for On-line Diagnosis of Distributed Systems," Research Report, 2008. [Online]. Available: http://hal.inria.fr/inria-00282294/en/ , [retrieved: 11, 2012]

[32] M. Shaw and D. Garlan, *Software architecture: perspectives on an emerging discipline*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

 ISBN: 978-1-61208-248-6