Integration of Semantic Web Services Principles in SOA to Solve EAI and ERP Scenarios
Towards Semantic Service Oriented Architecture

Tariq Mahmoud
Dpt. Business Informatics I
Carl von Ossietzky University
Oldenburg, Germany
mahmoud@wi-oldenburg.de

Jorge Marx Gómez
Dpt. Business Informatics I
Carl von Ossietzky University
Oldenburg, Germany
jorge.marx.gomez@uni-oldenburg.de

Abstract— Traditional EA concepts in the Enterprise Application Integration (EAI) domain focus on ex-post integration of application interfaces by pipelining different middleware technologies like message queuing or remote method invocations [1, 2]. Web Service enabled Service-Oriented Architectures (SOAs) used in the EAI context were a step towards providing an abstraction layer for the involved interfaces by using the Web Service Description Language (WSDL). The idea of SOA enlarges by applying Semantic Web Services (SWS) technology to it. OWL-S and WSMO are two examples for concrete implementation approaches in this area [3, 4], and WE will demonstrate semantic WSMO framework, and how it can solve the problems of traditional SOA.

Keywords- SOA, Semantic, EAI, SSOA, SWS, WSMO.

I. INTRODUCTION

It is obvious that EA developers have made a promise to find a solution to one of the major problems IT departments are facing - integration. The main idea from EA projects is to integrate processes via third-party applications and legacy systems to decrease the number of adapters needed developing when we have a connection between two systems. One of the reasons why a most of such those implementations fail is that the semantics of different systems have to be integrated at one point.

While EAI systems support a variety of ways to enable a correct syntactic integration by passing valid instance data, integrating the semantics of the underlying systems seems to be difficult. Another important idea is that there is a need to know the meaning of the low-level data structures in order to implement a semantically correct integration. But the problem is there is no formal definition of the interface data exists which implies that there is a need to consistency knowledge between the applications in integration project.

In this research we will involve Web Service enabled Service-Oriented Architecture (SOA) solutions based completely on standards, trying to decrease the integration effort. This new type of SOA depends on common Web Service technologies which allow interoperability by using standards like UDDI, WSDL, SOAP, etc. And as a consequence all important integration software providers from different backgrounds updated their products to support these Web Service Standards.

This new SOA model is based on the idea that business functionality is separated and published as services. Afterwards, they are used to be composed into a process.

This Web Service enabled SOA model has some fundamental advantages over traditional EA systems, and the gained advantages are:

- relying on standards provides a high degree of flexibility and offers an adaptable implementation;
- It becomes eventually possible to switch from a particular service to a different one without adaption;
- And the highly ability of reusing the functionality in a SOA.

According to this, a Web Service enabled SOA offers a solution to the standards problem by avoiding the central point of integration, which is often a bottleneck in previous EAI solutions. Also it reduces the number of point-to-point adapters because each interface is based on WSDL and it can communicate with each other WSDL enabled interface. What it doesn’t solve is the problem of making semantic documentation of such interfaces.

By extending the concept of SOA with semantics we will provide a formal description of the Web Service functionality allowing the developer to make the manual integration, if necessary at all, on this knowledge about the meaning of the data.

The Determining the semantics for interfaces means to make a definition of the concepts as well as the relationships between them through an ontology language like WSML. Also semantics offer the possibility of switching services dynamically by discovering them at runtime [5]. To make a close understanding of why we focus on semantic solution to Enterprise Architectures we must show the concepts of integrations in Enterprise Application Integration (EAI) and mention to the weakness of that system and then we will show how semantic solution can deal with those problems and find suitable solutions to them. So the next section explains the
concepts of EAI, and how Web Service enabled SOA establishes a different approach underlie current integration solutions.

II. MORALITY AND WEAKNESS OF EAI

In this paper, we used the term EAI to describe only the concept of semantic integration, but this term has also been used to name the systems itself because of the development which takes place in this area. And to illustrate how SSOA should solve EAI scenarios, we will first describe the functionality of these EAI systems in the following paragraph.

A. EAI Architecture Components

EAI solutions can be categorized into three basic layers that make up the majority of technologies common in today’s integration solutions. Each layer represents specific forms of EAI while also acting as services for solutions based on the layers above. These layers of abstraction form the building blocks of the EAI technology stack enabling the creation of more powerful solutions as you go further up the EAI stack. Making up the foundation of the EAI functional model is Core Middleware (Transport Layer). This layer represents the plumbing that allows multiple methods of application-to-application communications. Core middleware techniques and methods can be incorporated directly into the applications that need to communicate with one another. Core middleware, as the name implies, makes up the foundation with which most EAI solutions are built and includes database access routines, message-oriented-middleware (MOM), transaction processing middleware (TP), remote procedure calls, and distributed objects.

The message transformation and routing layer builds upon the core middleware foundation with a wide array of components that can further exploit the technologies mentioned above. This layer (Transformation Layer) includes tools to manipulate data contained in messages between applications. As an application generates a message, components in the message and transformation routing layer receive, re-view, revise, and reroute the message based on a set of rules predefined within the environment. By providing these services, applications do not need to include message queuing, data type matching, and application routing functionality. Instead, application developers can use the same mechanisms across all applications through the use of solutions that fall within the message transformation and routing layer. In addition to message brokers that are the common solution component in the layer, application servers have also begun to provide message transformation and routing services through message broker service components.

Most recent to emerge and extend the EAI functional model is the Business Process Management (BPM) Services Layer. This layer (Process Layer) contains tools and components that allow for the modeling of discrete business processes across multiple applications. By providing a layer of abstraction above applications themselves, BPM allows a modeler with little technical knowledge or expertise the ability to map out the appropriate movement of data across many applications, possibly at many different organizations, in order to complete business tasks. Although at its surface simple, the complexity resides in the set up and mapping of applications and procedures within the tools themselves which we will talk about it in this paper and describe the weakness in it and suggest some solutions. Appropriate communications channels and middleware components from the other two EAI layers must be in place before a BPM solution can be successfully implemented and used, Fig. 1 illustrate these EAI layers.

![The EAI Functional Model](image)

Figure 1. EAI Functionality.

And when we talk about EAI there is a difference between internal and external integration. Internal integration, often referred as intra-EAI, specifies the automated and event-driven exchange of information between various systems within a company. Another commonly used term for it is “Application to Application”-Integration (A2A). External integration, referred as inter-EAI, specifies the automated and event-driven information exchange of various systems between companies. In recent publications the second integration type is commonly referred to as “Business to Business”-Integration (B2B) [6]. And after the discussion we made above about the EAI layers, we can summarize and put some characteristics for each layer as follows [7]:

- **Process Layer**: Within this layer we can simply differ between components for process modeling and process representation support. The purpose of process modeling is to produce an abstraction of a process model called workflow type that can be either used for improved human understanding of operations within a specific domain or to serve as the basis for automated process representation [8]. In the case of EAI it is used for the latter. Process representation refers to the proactive control of the entire process from instancing a predefined workflow type, all the way, to its completion. And when talking about process models in EAI, there is a need to make a clear separation between private and public processes. Because this separation is the key to provide the necessary isolation and abstraction between the organization’s internal processes and processes across organizations.

- **Transformation Layer**: The Transformation layer ensures that the suitable routing and transformation of messages between the applications which will be integrated. Transformation refers to a process of selecting, targeting, converting and mapping data so
that it can be used by multiple systems. The transformation also addresses the data mismatch either at the lower-level of data type representation or at the higher-level of mismatched data structures. Mismatched data types may arise when two services use different binary representation for some data type. Dissimilar data structures on the other hand involve two different structures to represent the same body of data.

- **Transportation Layer**: This layer is responsible for the system- and platform- independent communication between the integration tool and the involved applications. It consists of a common protocol layer and adapters which transform external events in messages and vice versa. And we talked above about this layer, and we will let the details about this layer to network engineers to discuss it in further papers.

### B. Service Oriented EAI Architectures

Service Oriented Architecture (SOA) is an architectural style that guides all aspects of creating and using business processes, packaged as services, throughout their lifecycle, as well as defining and provisioning the IT infrastructure that allows different applications to exchange data and participate in business processes loosely coupled from the operating systems and programming languages underlying those applications. SOA represents a model in which functionality is decomposed into distinct units (services), which can be distributed over a network and can be combined together and reused to create business applications. These services communicate with each other by passing data from one service to another, or by coordinating an activity between two or more services. Although the concepts behind SOA were established before Web Services came along and a service within a SOA is completely independent of the concept of a Web Service, current SOA architectures employ them; Web Services naturally implement the philosophy of a SOA by using lightweight protocols based on widely accepted standards.

The difference between SOA approach and traditional EAI is primarily based on how the applications are seen in the architecture. The applications functionality is broken down into modules with well defined standardized interfaces. SOA treats services as a means of providing functionality in a whole business process. Applications have to provide the standardized interface in order to be integrated. If such an interface missed, it is still required to bind their functionality to a well-defined Web Service interfaces in order to make the transactions with other applications. This binding process creates an abstraction layer that hides all the complex details of the application similar to that achieved with traditional EAI adapters. But the difference exists in the standardization of the interface.

In concept, there are three main components in SOA architecture:

- **Service Provider**: The service provider creates a Web service and possibly publishes its interface and access information to the service registry.

- **Service Broker**: The service broker, also known as service registry, is responsible for making the Web service interface and implementation access information available to any potential service requestor categorized in taxonomies. The Universal Description Discovery and Integration (UDDI), defines the way to publish and discover information about Web services.

- **Service Requestor**: The service requestor or Web service client locates entries in the broker registry using various find operations and then binds to the service provider in order to invoke one of its Web services.

And fig. 2 illustrates the mechanism of publishing, discovering and binding web service in SOA environment:

![Figure 2. The activities in SOA.](image)

### C. SOA in EAI and its Weakness

In the following we will show how traditional SOA addresses the three functional layers of EAI and identify its limitations which are mainly caused by the absence of semantics in the service descriptions.

- **Process Layer**: The choice within SOA EAI solutions to implement private business processes is Workflow Management Systems (WfMSs) [8]. When WfMS are used with EAI, they have to allow the definition of workflow types and the execution of workflow instances as well. Traditional SOA implementations in the EAI domain apply to XML based Business Process Modeling Standards like BPEL, WSCI, and BPSS etc. Microsoft BizTalk is a famous SOA which incorporate with one of these standards, namely BPEL. And BPEL clearly separates private and public process models. It introduces the concept of executable process for private processes and abstract process for public processes.

From this scenario, is it possible at all to model the public processes? And the result exists in the fact that this definition is not executable, regardless of what standard is chosen in the process layer. And the reality is that there is incorporation between the public process in the execution of its private process [9]. According to this, the behavior of the public process of
two services offered by two different companies has to match in order to establish a communication. This leads to the necessity to define one process model for every partner the organization is conducting business with.

The second scenario is modeling the executable private process separately from the executable public process and making definition binding to relate both of them [9]. But, WfMS used within SOA does not have the concepts of dynamic binding of public and private processes. As Bussler describes: WfMS assume that workflow instances execute in isolation. If two workflow instances are related by the WfMS itself, one must be a subworkflow of the other one. Two top level workflow instances (in this case the private and the public process) can not be related by current WfMSs since no modeling concepts are available for this functionality [9].

- **Transformation Layer:** Starting from the point that SOA based on Web Services uses XML in defining the data structure of messages, transformation does not have to deal with the mismatches between structures. But in XML, the same data item could be modeled as an attribute of an existing element in one document and as a subelement in another XML representation. In current SOA document transformation becomes a very important mapping subprocess. Since current standards like XML and XML Schema only solve the mismatch on the syntactical and structural level; solving the mismatch on the semantic level is usually handled on a case-by-case basis. If, for example, two services use ebXML as their data structure, both trading partners have to transform ebXML to any kind of internal data structures they use. Transformation maps are used to handle and convert the content and structure of any source information based on its XML schema representation into any target document format. This solution requires a mapping between every two different XML schemas before an interaction between the corresponding Web Services can be set up. A widely used standard protocol for creating and saving these mappings is XSLT which is used to transform from one XML format to another.

- **Transportation Layer:** Since the protocol used for exchanging structured information in a Web Service enabled SOA is SOAP, it can easily be used with a variety of transport protocols such as HTTP, SMTP, and FTP.

SOA applying Web Services assumes that WSDL is a compliant interface of the participating applications. If the application does not support a WSDL interface, adapters will be used to transform external events into messages and vice versa, and this integration is not further discussed in this paper.

III. TOWARDS SEMANTIC SOA

Although traditional Web Services can simplify a SOA drastically, the technologies are exclusively syntactical that lack the consideration of semantics. With the use of Semantic Web markup languages data structures passed through Web Service interfaces are expressed in ontologies creating a distributed knowledge base.

It becomes a lot easier to understand what the service actually can be used for. And this provides the means for agents to reason about the web service description and to perform automatic Web service discovery and execution.

A. A Brief Look on Semantic Framework, WSMF

WSMF [5] consists of four different main elements for describing semantic web services (see Fig. 3): ontologies that provide the terminology used by other elements, goals that define the problems that should be solved by web services, web services descriptions that define various aspects of a web service, and mediators which bypass interpretability problems.

WSMO is a formal ontology for describing various aspects related to Semantic Web Services. And WSMO is based on the Web Service Modeling Framework [5]. The objective of WSMO is to define a consistent technology for Semantic Web Services by providing the means for semi-automated discovery, composition and execution of Web Services which are based on logical inference-mechanisms. WSMO applies WSML as the underlying language based on different logical formalisms.

WSMO defines four main modeling elements for describing several aspects of SWS:

- **Ontologies:** They represent the key element in WSMO, firstly to define the information’s formal semantics and secondly to link machine and human terminologies. Ontology is a formal explicit specification of a shared conceptualization [Gruber, 1993] WSMO ontologies give meaning to the other elements (Web Services, goals and mediators), and provide common semantics, understandable by all the involved entities (both humans and machines).
• Goals: Goals represent the objectives of the service requester to be accomplished when consulting a Web Service. They provide the means to express a high level description of a concrete task. A goal can import existing ontologies to make use of concepts and relations defined somewhere else, either extending or simply reusing them. The advantage of using goals is that requesters only have to provide a declarative specification of what they want. It is not necessary for the requester to have a prior knowledge of the Web Service or to browse through a UDDI registry to find services providing the appropriate functionality.

• Web Services: Similar to the way requester declares their goals, every Web Service capability has to be declared. Additionally the interface, used mediators and non functional properties have to be defined. Only if the service requester and provider use the same ontology in their respective service description, the matching between the goal and the capability can be directly established. Unfortunately, in most cases the ontologies will differ and the equivalence between a goal and a capability can only be determined if a third party is consulted for determining the similarities between the two ontologies. For this, WSMO introduces another modeling element: the mediators.

• Mediators: The current specification contains four different types of mediators: ooMediators, ggMediators, wwMediators and wgMediators.
  - ooMediators: mediators that have the role of resolving possible representation mismatches between ontologies.
  - ggMediators mediators that have the role of linking two goals. This link represents the refinement of the source goal into the target goal or states equivalence if both goals are substitutable.
  - wwMediators mediators that link Web Services to goals, meaning that the Web Service can fulfill the goal to which it is linked.
  - wgMediators mediators that are used for linking two Web Services in the context of automatic composition of Web Services.

B. Proposed Semantic SOA Concepts

In semantically enabled a SOA there is a need to redefine the three main concepts of the traditional SOA (see paragraph II.B) in this way:

• Service Provider: WSDL can still be in use as a universally accepted interface language, but additionally there is a need to provide a WSMO compliant service description (see paragraph 3.1). By doing so requester will be allowed to discover the service based on a formally defined goal instead of searching through the directory service and selecting the suitable service.

• Service Broker: The functionality of the service provider (registry) remains the same. The only difference between it and traditional SOA service broker is that it stores WSML service descriptions instead of WSDL descriptions.

• Service Requester: Service requesters have to publish the desired functionality as a WSMO goal instead of the traditional way in SOA.

C. SSOA Virtue over SOA Weakness

We will speak in the following paragraph how SSOA overcomes the limitations of traditional SOA depending on the discussion on the layers of EAI architecture which we described it before (see paragraph II.A.):

• Process Layer: We mentioned previously that different services require different message exchange patterns in order to have a consistent state transition within the service. And the problem here which needs to solve is that two different services may provide different interfaces (public process) for the semantical same operation [9]. For example when we make purchasing for a product, one system might offer it with one single invocation while other system requires first the process of creation a user, and then activation of the user and finally the purchase of the product. Therefore, in the first system, one invocation completely defines a user whereas in the second system several interface invocations are necessary to achieve the same functionality. More important than the invocation is its specific execution order so: Activation of the user cannot be achieved before the creation of this user. This difference in the public processes can cause heterogeneity after successful discovery of a Web Service within a SOA. In order to invoke the Service, the two participants must be able to redefine their communication patterns, or have to use an external mediation system as a part of the process. The first solution implies changes in the requester’s or provider's business logic (private processes) and prevents the dynamic invocation. Every party has to modify its pattern for every invocation of Web Services with different communication patterns. The second is done through an approach called process or behavior mediation [5]. By applying SWS principles mediator systems can compensate the clients, communication pattern or the Web Services communication pattern, in order to obtain equivalent public processes. The role of mediators is putting the necessary means together for the runtime analyses of two given instances of a public process, and compensating the possible mismatches that may appear. For example, to group several messages in a single message, we need to change their order, or even to remove some of the original messages in order to facilitate the communication between the two parties to get the unified model for that system.

• Transformation Layer: Although ontologies are used as shared conceptualizations of the same problem domain
within SWS, it will always be such a case in which there are ontologies for the same domain created by different vendors around the world. In this case where services use dissimilar ontologies, the EAI solution must provide the means to transform between the different ontologies. This semantic transformation is called mediation [5]. Mediation ensures that the semantics of the involved concepts are guarded. Similar to the transformation in traditional EAI solutions it is still impossible to use a mapping tool applied on ontologies to automatically generate these mapping rules. Some of the best results of research projects in this area are mapping tools that which are able to validate or to suggest possible mappings, but at some point in the mapping process, there is still a need to domain expert intervention. Anyway, the main difference is that the mediation takes place at the level of ontologies rather than on the XML Schema level as with traditional transformation in SOA. This implies that for every two different XML Schemas, mapping rules have to be defined, whereas for two messages using different ontologies only a mediator (mapping) between the two ontologies have to be in place. This sounds similar in the first place, but XML Schemas are defined for every message, ontologies on the other hand define a greater domain. Furthermore the mappings between two different ontologies can be discovered and reused at runtime, whereas XML Schema mappings have to be bound at the design time. Similar to the mapping between different Schema representations of XML messages (e.g. between DTD and XML Schema), ontology mediation has to be applied to the conceptual model of the involved ontology representations as well (e.g. between WSMO language variants and OWLS language variants). However, this transformation problem has a restricted nature since the set of available ontology representation languages is limited.

**CONCLUSION & FUTURE WORK**

In this paper we showed the new term: Semantic Service Oriented Architecture, and explain a semantic framework, WSMO to the SSOA.

In this paper we had talked in general about EAI architecture and we tried to describe how this SSOA new architecture could solve EAI scenarios. And in the future work, we will try to implement a semantic framework like WSMO on FERP [10], Federated ERP system, to make a semantic web service enabled system.

**REFERENCES**