Exposing Semantic Web Service principles in SOA to solve EAI scenarios

Exposing Semantic Web Service principles in SOA to solve EAI scenarios

Armin Haller
Digital Enterprise Research Institute
National University of Ireland
Galway, Ireland
armin.haller@deri.org

Juan Miguel Gomez
Digital Enterprise Research Institute
National University of Ireland
Galway, Ireland
juan.gomez@deri.org

Christoph Bussler
Digital Enterprise Research Institute
National University of Ireland
Galway, Ireland
christoph.bussler@deri.org

ABSTRACT

Traditional Enterprise Application Integration (EAI) focuses on the integration of application interfaces by pipelining different middleware technologies like message queuing or remote method invocations. Web Service enabled Service-Oriented Architectures (SOAs) used in EAI were a step towards providing an abstraction layer for the involved interfaces by using the Web Service Description Language (WSDL) [9]. We enlarge the notion of SOA by applying Semantic Web Services (SWS) technology to it. The architecture employs principles developed in work already published on SWS architectures [18, 40] to demonstrate their applicability in EAI. We examine what current SWS technology offers in respect to the requirements imposed by EAI scenarios. The major focus of the paper is to point out the potentials current SWS technology offer for EAI. This analysis includes some challenges for SWS frameworks to fully enable dynamic discovery and invocation in EAI. It further concludes that both are already possible in intra-EAI scenarios under certain assumptions. We demonstrate the applicability of the Web Service Execution Engine (WSMX), as first kind of a Semantic Service Oriented Architecture (SSOA) in a high-level use-case to exemplify the activities within the lifecycle of such an architecture.

1. INTRODUCTION

In recent years Enterprise Application Integration (EAI) systems have evolved promising to solve one of the major problems IT departments are facing - integration.

The business driver behind EAI projects is to integrate processes across third-party applications as well as legacy systems to decrease the number of adapters one has to develop if connecting two systems [29]. One of the reasons why a majority of EAI implementations fail (some articles even account for 70% of EAI projects as failure [21]) 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 through passing valid instance data [33], integrating the semantics of the underlying systems turns out to be difficult. Engineers integrating the enterprise application systems have to know the meaning of the low-level data structures in order to implement a semantical correct integration. No formal definition of the interface data exists [8] which implies that the knowledge of every developer of applications involved in the integration project is assumed to be consistent.

Another problem early EAI systems faced laid in their origin. They consisted mainly of a message-oriented middleware like WebSphere MQ (formerly MQSeries)[1], adapters and connectors to access source and target components, transformation engines to map information formats between source and target systems and some intelligent routing [13]. Standards for these components, mainly the internal semantics of the EAI system the adapter frameworks mapped to, the transformation maps themselves, the process definitions etc. were not considered at all in the beginning. Only when the EAI market grew did the issue of standards start to appear. Proprietary EAI vendors with market strength were keen to declare their own implementations as standards. End users demanded standards to avoid the vendor lock-in and subsequently, to avoid a problem, they initially tried to solve with the EAI solution - to be able to integrate their EAI system with other EAI systems [13]. Finally new entrants entered the market advertising Web Service enabled Service-Oriented Architecture (SOA) solutions based completely on standards, claiming to make the integration effort flawlessly.

This new type of SOA relies on common Web Service technologies which allow interoperability by relying on standards like UDDI [11], WSDL [9], SOAP [20] etc. As a consequence all major integration software providers from different backgrounds (Application Server vendors, EAI vendors, Portal software vendors etc.) enlarged their products to support these Web Service Standards [33]. Eventually some of them (examples include IONA Artic[2] and Microsoft BizTalk[3] even redeveloped their integration suite to realise the integration as SOA.

This new SOA model is based on the principle that business functionality is separated and published as self-contained components, called services. They are subsequently used to be composed into a process. This Web Service enabled SOA model has some fundamental advantages over traditional EAI systems:

- the functionality in a SOA can be reused to a high degree;
- relying on standards it provides a highly flexible and adaptable implementation;
- and it becomes eventually possible to switch from a particular service to a different one without adaptions.

According to this, a Web Service enabled SOA offers a solution to the standards problem by avoiding the central point of integration, often a bottleneck in EAI solutions. Furthermore it still reduces the number of point-to-point adapters since every interface is based on WSDL and can communicate with every other WSDL enabled interface. What it does not solve is the problem of documenting the semantics of these interfaces.

[1]see http://www-306.ibm.com/software/integration/wmq/
[2]see http://www.iona.com/products/artix/
[3]see http://www.microsoft.com/biztalk/

WWW2005, May 10–14, 2005, Chiba, Japan.
.

By enlarging the notion of SOA with semantics we provide a formal description of the functionality of a Web Service to allow the developer to base the manual integration, if necessary at all, on this knowledge about the meaning of the data. Determining the semantics for interfaces means to define the concepts as well as the relationships between them through an ontology language [18]. Semantics bring closer the possibility of switching services dynamically by discovering them at runtime.

The next section explains the concepts of EAI, how Web Service enabled SOA constitutes a different approach and what limitations still underlie current integration solutions. In section 3.1 we describe WSMO as the underlying framework for our Semantic Service-Oriented Architecture (SSOA). The approach followed to apply Semantic Web Services to SOA is illustrated in detail in section 3.3. In the same section we explain what benefits a SSOA offers in EAI scenarios. Furthermore we point out some remaining challenges on SWS frameworks to exploit a fully dynamic environment. Section 4 showcases the applicability of the Web Service Execution Environment (WSMX) as a SSOA in EAI scenarios and illustrates it on a use case. The paper concludes with a short discussion.

2. VIRTUE AND LIMITATIONS OF EAI

In the original sense the term EAI described only the concept of semantic integration, but since the evolvement of systems covering this domain the term has also been used to name the systems itself. To subsequently show how semantically enabled SOA are applied in EAI scenarios we first depict the functionality constituting these EAI systems in the following paragraph.

2.1 EAI Concepts and Functionality

We can differ 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) [7]. 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 B2B integration [7].

EAI systems provide different types of integration levels explaining the various dimensions of the integration tasks. Literature identifies the following three layers common to EAI systems [43, 14].

Process Layer

Within the process layer we differ between components for process modelling and process enactment support.

The purpose of process modeling is to produce an abstraction of a process model called workflow type [22] that can be either used for improved human understanding of operations within a domain or to serve as the basis for automated process enactment. In the case of EAI it is used for the latter.

Process enactment refers to the proactive control of the entire process from instancing a predefined workflow type all the way to its completion.

When talking about process models in EAI a clear separation between private and public processes has to be made. This separation is well established in research [6] and different business process modelling standards (i.e. BPEL [1], WSCI [2]) incorporate it. It is the key to provide the necessary isolation and abstraction between the organisation's internal processes and processes across organisations.

Transformation Layer

This layer ensures the proper routing and transformation of messages between the applications to be integrated. Transformation refers to a process of selecting, targeting, converting and mapping data so that it can be used by multiple systems [43]. The transformation addresses the mismatch of data 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

The Transportation Layer provides the system- and platform-independent communication between the integration tool and the participating applications. It consists of a common protocol layer and adapters which transform external events in messages and vice versa [43].

Most EAI systems support some sort of asynchronous transport layer, either proprietary or open ones. For instance, many leverage IBM WebSphere MQ or more open systems such as Java Messaging Service (JMS) or the Object Messaging System (OMS) [33].

2.2 Service-Oriented EAI Architectures

SOA is a set of independently running services communicating with each other in a loosely coupled manner via event-driven messages. 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 [24]; Web Services naturally implement the philosophy of a SOA by using lightweight protocols based on widely accepted standards.

Established technologies for SOA include for example the Common Object Request Broker Architecture (CORBA)[4] administered by the Object Management Group (OMG). It is a vendor-independent architecture that applications based on different programming languages can use to work together over networks. In contrast to Web Services which are mostly based on SOAP/HTTP, services in CORBA typically communicate via the IIOP (Internet Inter-ORB Protocol). The second major differences is the tight coupling of two CORBA services in comparison to the loose coupling between Web Services. In CORBA objects are shared between components, whereas Web Services communicate primarily over message.

The difference in the SOA approach to traditional EAI lies primarily in the way applications are seen in the architecture. The applications functionality is broken down into modules with well defined standardised interfaces. SOA treats services as a means of providing one functionality in a whole business process. Applications have to provide the standardised interface themselves in order to be integrated. If they lack such an interface, it is still required to wrap their functionality to a well-defined Web Service interfaces in order to participate in interactions with other applications. This wrapping process creates an abstraction layer that hides all the complex details of the application similar to that achieved with traditional EAI adapters. The difference lies in the standardisation of the interface.

At a conceptual level, SOA is composed of three core pieces [26]:

[4]see http://www.corba.org/

- **Service registry**: It acts as an intermediary between providers and requesters. Most of these directory services categorise services in taxonomies.
- **Service provider**: The Service Provider defines a service description and publishes it to the service registry.
- **Service requester**: The service requester can use the directory services' search capabilities to find service descriptions and their respective providers.

The three activities the service requester and provider in a SOA as depicted in figure 1 can perform are [26]:

- **Publish:** The service provider has to publish the service description in order to allow the requester to find it. Where it is published depends on the architecture.
- **Discover:** In the discovery the service requester retrieves a service description directly or queries the service registry for the type of service required.
- **Invoke:** In this step the service requester invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact and invoke the service.

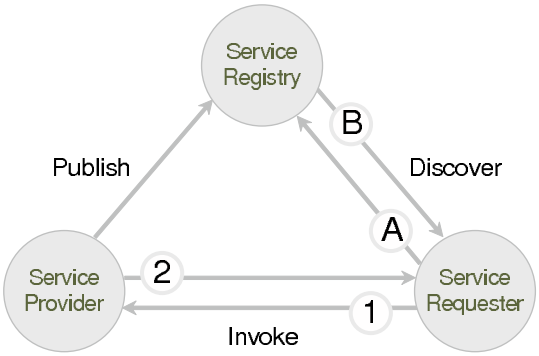

Figure 1: Activities in a SOA cp. [25]

2.3 SOA in EAI and its weakness

In the following we 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 technology of choice within SOA EAI solutions to implement private business processes are Workflow Management Systems (WfMSs) [22]. As mentioned above WfMSs used for EAI have to allow both the definition of workflow types and the execution of workflow instances. Traditional SOA implementations in the EAI domain apply to a growing extent XML based Business Process Modelling Standards like BPEL [1], WSCI [2], BPSS [10] etc. Microsoft BizTalk is a prominent SOA incorporating one of these standards, namely BPEL. BPEL clearly separates private and public process models. It introduces the concept of executable process for private processes and abstract process for public processes.

If it is possible at all to model the public processes, this definition is not executable, regardless of what standard is chosen in the process layer. One has to incorporate the public process in the execution of its private process [6]. The behaviour of the public process of two services exposed by different companies have to match to establish a communication. This leads to the necessity to define one process model for every partner the organisation is conducting business with.

An alternative approach is to model the executable private process separately from the executable public process and define binding to relate both [6]. However, WfMS used within SOA does not have the concepts of dynamic binding of public and private processes. As Bussler [6] describes, WfMS assume that workflow instances execute in isolation. Two top level workflow instances (in this case the private and the public process) can not be related by current WfMSs since no modelling concepts are available for this functionality.

Transformation Layer

Since SOA based on Web Services use XML as their way of defining the data structure of messages, transformation does not have to deal with mismatches between e.g. hierarchical structures or fixed-length fields. However in XML the same data item may be modelled 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 functionally exposed 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 RosettaNet as their data structure, both trading partners have to transform RosettaNet to whatever internal data structures they use. Transformation maps are used to process and convert the content and structure of any source information based on its XML schema representation into any target document format [45]. This solution requires a mapping between every two different XML schemas before an interaction between the respective Web Services can be set up. A widely used standard protocol for creating and saving this mappings is XSLT.

Transportation Layer

SOAP as the protocol used for exchanging structured information in a Web Service enabled SOA can be used with a variety of transport protocols such as HTTP, SMTP, and FTP.

SOA applying Web Services assumes a WSDL compliant interface of the participating applications. If the application does not natively support a WSDL interface, adapters are used to transform external events into messages and vice versa.

Both the transport protocol and the adapter integration is not further discussed.

3. SWS AS FOUNDATION FOR EAI

As mentioned above although traditional Web Services can simplify a SOA drastically, the technologies are exclusively syntactical that lack the consideration of semantics. In the following we show a simple artificial Web Service, which allows the user to register a child with the Austrian record section. Figure 2 shows how the respective WSDL description could look like. It is evident that only humans can interpret the meaning of e.g. ”hasBirthdate” and what the allowed vocabulary for the String ”hasBirthplace” would be.

With the use of Semantic Web markup languages data structures passed through Web Service interfaces are expressed in ontologies

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Namespace Defintion --> [...]

  <message name="ChildInput">
    <part name="hasBirthdate" type="xsd:date"/>
    <part name="hasBirthplace" type="xsd: string "/>
  </message>
  <message name="CitizenshipRespone">
    <part name="hasCitizenship" type="xsd: string "/>
  </message>

  <portType name="CitizenshipPortType">
    <operation name="GetCitizenship">
      <input message="tns:ChildInput"/>
      <output message="tns:CitizenshipRespone"/>
    </operation>
  </portType>

<!-- Binding Definition --> [...]
```

Figure 2: Example WSDL service description

createing a distributed knowledge base. It becomes a lot easier to comprehend what the service actually can be used for. This provides the means for agents to reason about the web service description and to perform automatic Web service discovery and execution [32]. In the following we describe the Web Service Modeling Ontology (WSMO) and how the same service described in figure 2 could be expressed in the Web Service Modeling Language (WSML).

3.1 WSMO, a SWS Framework

WSMO [42] is a formal ontology for describing various aspects related to Semantic Web Services. WSMO is based on the Web Service Modeling Framework [18]. The objective of WSMO and its surrounding efforts is to define a coherent technology for Semantic Web Services [32] by providing the means for semi-automated discovery, composition and execution of Web Services which are based on logical inference-mechanisms. WSMO applies WSML [16] as the underlying language based on different logical formalisms. WSMO defines four main modelling elements for describing several aspects of SWS [42]:

- **Ontologies:** They represent the key element in WSMO, firstly to define the information's formal semantics and secondly to link machine and human terminologies (see figure 3 which shows a section of a sample ontology used for the capability description below). 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 fulfilled 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 elsewhere, 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 them to have a prior knowledge of the Web Service or to browse through a UDDI registry to find services providing the appropriate functionality.

```
ontology
 <!-- Non Functional Properties Definition --> [...]
 <!-- Used Mediators Definition --> [...]
 <!-- Imported Ontologies Definition --> [...]

 concept Human
  nonFunctionalProperties
   dc# description hasValue "concept of a human being"
  endNonFunctionalProperties
  hasName ofType foaf#name
  hasParent inverseOf(hasChild) impliesType Human
  hasChild impliesType Human
  hasAncestor transitive impliesType Human
  hasWeight ofType (1) xsd#decimal
  hasWeightInKG ofType (1) xsd#decimal
  hasBirthdate ofType (1) xsd#date
  hasObit ofType (0 1) xsd#date
  hasBirthplace ofType (1) loc#location
  isMarriedTo symmetric impliesType (0 1) Human
  hasCitizenship ofType oo#country

 concept Child subConceptOf Human
  nonFunctionalProperties
   dc# relation hasValue {ChildDef, ValidChild}
  endNonFunctionalProperties

 axiom ChildDef
  NonFunctionalProperties
   dc# description hasValue "Human being not older than 12
       (the concrete age is an arbitrary choice)"
  endNonFunctionalProperties
  definedBy
    forall ?x (
     ?x memberOf Human and
     ?x[hasAgeInYears hasValue ?age] and ?age=<12
     implies ?x memberOf Child).

 [...]
```

Figure 3: Example WSMO ontology definition (cp. [18] for the whole ontology)

- **Web Services:** Similar to the way requester declare their goals, every Web Service capability has to be declared (see figure 4 for a sample capability description expressing the same functionality as the WSDL listing above). 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 modelling element: the mediators.

- **Mediators:** The current specification contains four different types of mediators: ooMediators, ggMediators, wwMediators and wgMediators. The ooMediators have the role of resolving possible representation mismatches between ontologies. The ggMediators have the role of linking two goals. This link represents the refinement of a source goal into a target goal. If two goals are equivalent, then a Web Service that can satisfy one of them can satisfy the other one as well. The wgMediators link Web Services to goals, meaning that the Web Service can fulfill the goal to which it is linked. The

```
webService "http :// example.org/ BirthRegistrationAustria "
 <!-- Non Functional Properties Definition --> [...]
 <!-- Used Mediators Definition --> [...]
 <!-- Imported Ontologies Definition --> [...]

 capability
   precondition
      nonFunctionalProperties
       dc# description hasValue "The input has to be boy or a girl
         with birthdate in the past and the birthplace Austria."
      endNonFunctionalProperties
      definedBy
        (? child memberOf Child)
          and ? child [ hasBirthdate hasValue ? birthdate ]
          and wsml#dateLessThan(?birthdate, wsml#currentDate())
          and ((? child [ hasBirthplace hasValue ? location ]
          and ? location [ locatedIn hasValue loc#at ])).

   assumption
      nonFunctionalProperties
        dc# description hasValue "The child is not dead"
      endNonFunctionalProperties
      definedBy
        (? child memberOf Child)
          and neg ( exists ?x (? child [ hasObit hasValue ?x ])).

   effect
      nonFunctionalProperties
        dc# description hasValue "After the registration the child
          is an Austrian citizen"
      endNonFunctionalProperties
      definedBy
        ? child memberOf Child
          and ? child [ hasCitizenship hasValue loc#at ]

 <!-- Interface Definition --> [...]
```

Figure 4: Example WSMO capability description

wwMediators are used for linking two Web Services in the context of automatic composition of Web Services.

3.2 Towards a semantically enriched SOA

We apply the framework proposed by the Web Service Modelling Ontology (WSMO) to the concept of a SOA and call this new type of architecture Semantic Service-Oriented Architecture (SSOA). This SSOA takes the Conceptual Semantic Web Service architecture of Fensel/Bussler [18] and Preist [40] as an input. Hence the aim of this paper is not to design yet another Semantic Web Service architecture, but to show how one kind of such an architecture performs in the EAI domain.

By semantically enriching a SOA we have to redefine the three main concepts of the SOA (see section 2.2) in the following way.

- **Service provider** They can still use WSDL as a universally accepted interface language, but additionally they have to provide a WSMO compliant service description (see section 3.1). By doing so they allow a requester to discover the service based on a formally defined goal instead of searching through the directory service and selecting the appropriate service.
- **Service registry** The functionality of the service registry remains the same. The only difference is that it stores WSML service descriptions instead of WSDL.
- **Service requester** Service requesters have to publish the desired functionality as a WSMO goal. In case of a SSOA the requester can also be an agent requiring one of a class of abstract service. If so the requesting agent has to have its own interface defined in WSML (to allow WSMX to mediate between possible differences in the message exchange patterns of the requester and provider).

We adopt in the following the Business-to-Business E-Commerce lifecycle model defined in [47] to help us understand the activities to be considered to discover and invoke a service in a SSOA. Since the third aspect, the Publish operation (see section 2.2) has to happen at design time, it is not included in this lifecycle denoting the dynamic character of a SSOA.

1. Matchmaking: The first activity in the matchmaking phase is the Web Service Discovery. It deals with the matching of formalised goals and formalised Web Service descriptions. In this step it has to be proven that the capability logically entails the goal with the premise that the conditions for successful usage of the Web Service are fulfilled [48]. The notion of matching goals to services is similar to component matchmaking, cf. [30, 37]. The matchmaking includes the following three subactivities:
   - Ontology Mediation: To be able to match Web Service descriptions with goals defined in a different ontology, ontology mediators are called if available to resolve possible differences.
   - Handling Partial Matches: Goal-Capability matching is only successful, if the goal and the Web Service capability match perfectly. This does not hold for many cases, as there might be semantic differences between the goal and the capability. Nevertheless a Web Service might be usable for solving a goal even if some part of the goal is in conflict with the service description examined. Several degrees of matches can be considered, each varying in the degree of satisfaction of the user's goal. Colucci et al present an approach how to calculate the degree of this partial matches in [12].
   - Service discovery [48] uses the Web Services matched in the previous step to access the real services behind such Web Service interfaces, finally checking what services fulfill the requester's goal. The Web Service Discovery results in an agreed service [40] defining what services of interest to the requester the provider can offer. The difference between the aforementioned Web Service discovery and the Service Discovery can be easiest described with an example. Web Service discovery is to find a Web Service that can be used to e.g. purchase automotive parts whereas service discovery is about checking whether the actual Web Services can really provide the concrete requested part.
2. Filtering results: The outcome of the previous phase is a set of suitable Web Services. In order to improve the quality of the result set and to ultimately choose one, additional filter mechanism can be applied in the selection. They can be based on non-functional properties of Web Services or take some service requesters preferences into account.
3. Agreement Negotiation: After selecting a Web Service two parties need to determine if they can agree on a service which the provider will supply to the requester. The agents enter into negotiation with each other, to see if they can agree mutually acceptable terms of business.

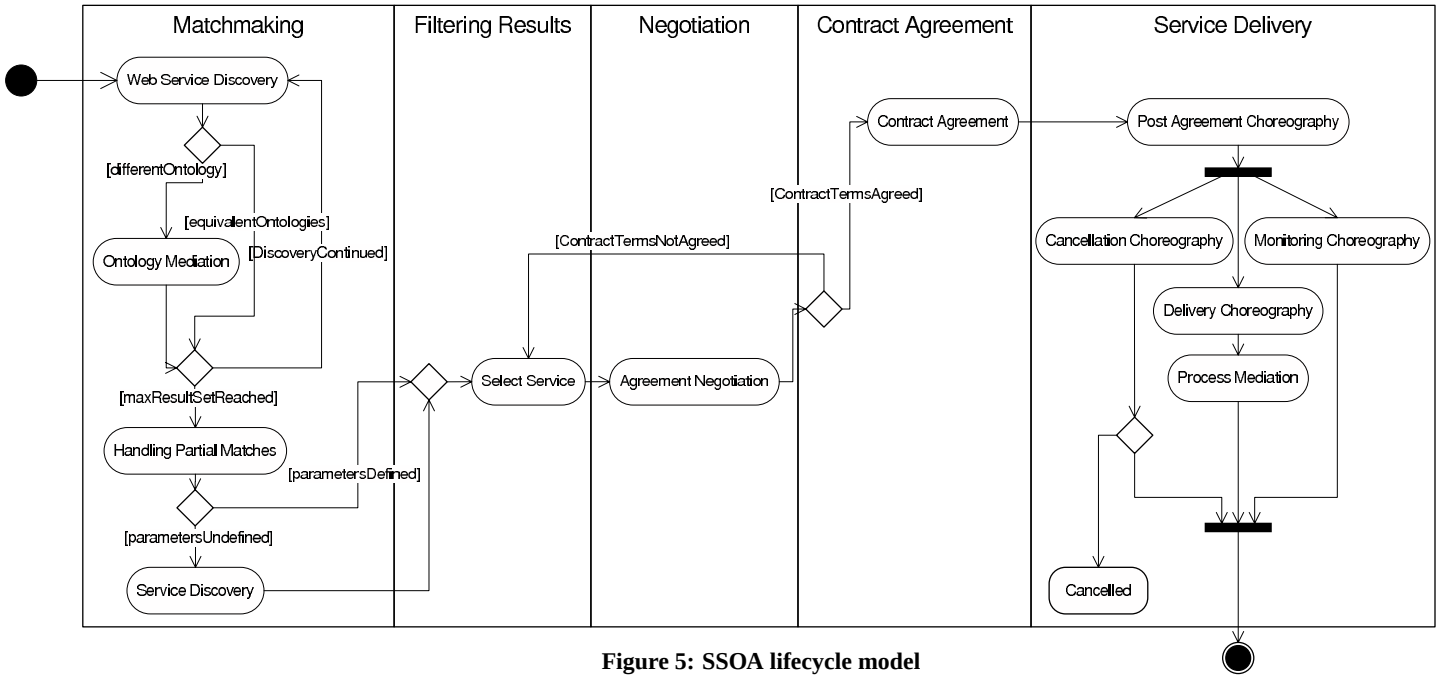

Figure 5: SSOA lifecycle model

4. Contract Agreement: The outcome of this phase is a legally binding contract, specifying the terms that both parties consider acceptable.

5. Service Delivery: The agents carry out the agreed transaction, within the parameters specified in the contract. The service delivery contains a post-agreement choreography which includes the following activities [40]:

   - Delivery Choreography: This governs the actual message exchange pattern which controls the execution of one or more tasks within the service delivery. If the requester's and provider's communication patterns differ, behaviour mediation would try to resolve them at this stage of the lifecycle.
   - Monitoring Choreography: This does not directly control or influence the delivery choreography itself, but it allows the requester to monitor its state. For example it would allow the requester agent to obtain the delivery status of a purchase.
   - Cancellation/Renegotiation Choreography: This Choreography defines if a cancellation or alteration of a service is allowed and within which circumstances.

3.3 How SSOA overcomes limitations of traditional SOA

Process Layer

As mentioned above different services require different message exchange patterns (i.e. which interface has to be invoked when) in order to achieve a consistent state transition within the service. The problem to solve is that two different services might provide different interfaces (public process) for the semantical same operation [8]. If you take an example of purchasing a book, one system might offer it with one single invocation whereas the other system requires first the creation of a user, followed by activation of the user and finally the purchase of the book. Hence in the first case, one invocation completely defines a user whereas in the second case several interface invocations are necessary to achieve the same functionality. More important than the mere invocation is its specific execution order: Activation of the customer cannot be before the user is actually created. 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 parties must be able to redefine their communication patterns or to use an external mediation system as part of the process. The first solution implies changes in the requester's or providers business logic (private processes) and hinders a dynamic invocation. Every participant would have to readjust its pattern for every invocation of Web Services with heterogeneous communication patterns.

The second is done through an approach called process or behaviour mediation [18]. 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 to put together the necessary means for the runtime analyses of two given instances of a public process and to compensate the possible mismatches that may appear. For instance, to generate dummy acknowledgement messages, to group several messages in a single one, to change their order or even to remove some of the messages in order to facilitate the communication between the two parties.

Transformation Layer

Although ontologies are used as a shared conceptualizations of the same problem domain within SWS it will always be the case that there are ontologies for the same domain created by different entities around the world. In such a case where services use dissimilar ontologies the EAI solution has to provide the means to transform between them. This semantic transformation is also called mediation [18]. Mediation ensures that the semantics of the involved concepts are preserved.

Similar to the transformation in traditional EAI solutions it is still impossible for 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 are able to validate or to suggest possible mappings, but at some point in the mapping process, domain expert intervention still remains a necessity [34]. However 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. Precisely this entails 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 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 OWL-S language variants) [34]. However, these transformation problem is of restricted nature since the set of available ontology representation languages is limited.

3.4 Prerequisites of SWS in inter-EAI

As explained above SWS principles enable to dynamically discover and invoke services within a SOA. Furthermore they facilitate the functional requirements within EAI on any necessary mediation based on data and process ontologies and on-the-fly translation of their concepts into each other.

However, despite the conceptual advantages SWS technology offers for EAI scenarios, due to the infancy of the frameworks itself, work in the following fields have to advance in order to be capable to achieve the ultimate goal of on-demand consumption of services. In the following we describe the challenges associated with the individual activities in the above defined lifecycle model.

Matchmaking

Non-functional properties

In Business-to-Business (B2B) scenarios it is of utmost importance that a provided service is legally binding once it is invoked. This is achieved in traditional B2B integration scenarios via Service Level Agreements (SLAs). A SLA constitutes a contract between a service provider and requester specifying the level of service that is expected during its term [28]. They can specify availability (refers to the temporal and locative properties when a service can be requested or provided), response times, rights and economic compensation if the invocation of the service fails or the actual real world service is not conducted, etc.

It is important to note that the provider's formal service description represents an agreement about the Web Service functionality, but it does not need to have any legal status. In the case of dynamic discovery and invocation the above mentioned parameters of a SLA have to be included in the service description in order to be accounted for in the Matchmaking phase.

O'Sullivan et al [36] define the notion of obligations in a recent technical report in order to capture the responsibilities of both the service provider and service requester. By defining non-functional properties they are available in the filtering phase to resolve the result set to the services compliant to the defined obligations. For example, a service provider may wish to advertise that service requesters have an obligation for a relationship if they request their service.

These obligations to define a SLA are currently not addressed in the SWS frameworks and it remains a challenge to incorporate support for richer non-functional properties to reflect constraints on properties such as payment, availability, rights and economic compensation.

Ontologies for Business Documents

Messages in B2B interaction are sophisticated pieces of information and several organizations tried to standardize common business documents like purchase orders etc. These standards are called B2B protocol standards [5]. A B2B protocol standard in general is not only the description of the message formats exchanged (e.g. purchase order), but also properties like bindings to transport protocols, the sequencing, the security and many more.

The most well-known standards in regard to the message format are EDI[5] and SWIFT[6]. They not only provide a defined syntax but also a defined vocabulary for values of the message fields. However these standards are not applicable in SOA. They are neither XML based, nor is it easy to set up an EDI or SWIFT compliant interaction. Since these standards maintained great flexibility in defining the involved messages, it requires the two parties to agree on interaction protocols and message formats.

RosettaNet[7], ebXML[8], OAGIS[9] and many others are industrial consortiums which aim to ease this process by using XML and XML Schema technologies to define standardised syntax of messages. They also constitutes the standards supported in traditional Web Service enabled SOA. Again, some of these standards are merely defining the message format others are defining the transport protocol, the sequencing, etc. By using one of these standards, organisations do not need to go through an agreement phase to specify the used vocabulary for values of the message fields. Instead it is sufficient to agree on what standard to use. However, these standards have necessarily maintained some flexibility in defining the message format. Since they are syntactic, rather than semantic it can not be checked in the discovery of a service if the expected message format is compliant to the provided message format.

By applying ontologies in describing business documents in a SSOA the matchmaking becomes possible even if two messages differ in the used concepts and attributes. As mentioned above, mediation is necessary if different ontologies are used. The challenge for SWS frameworks is to set up or initiate standard ontologies for business documents. As long as every service requester or provider is using its own conceptualisation of the same domain a matchmaking between

[5] see http://www.x12.org
[6] see http://www.swift.com
[7] see http://www.rosettanet.org
[8] see http://www.ebxml.org
[9] see http://www.openapplications.org

the goal and the service description is difficult or even impossible. Proposals for ontologizing business documents are made in the recent past [19], but an agreement over such ontologies have to be achieved over vertical markets.

Policies as logical conditions

Since most B2B integration applications have fundamental requirements around security and reliability as a kind of minimum criterion for adoption, most add on standards for the Web Service Stack were positioned around security and reliability. These Standards have reached a relative mature status and are already implemented in some Web Service integration platforms [35].

Since participating entities in a Web Service interaction rely on the ability for message processing intermediaries, traditional point-to-point security mechanisms using a secure transport (e.g. SSL/TLS) are not applicable [35]. Specifically, the SOAP message model operates on logical endpoints that abstract from the physical network and application infrastructure and therefore frequently incorporates a multi-hop topology with intermediate actors.

The proposed Web Service security standards provide a mechanism for end-to-end security. The WS-Security [3] standard defines among other things how to attach and include security tokens within SOAP messages. The requester's security token can either be directly known and trusted by the service provider or either the requester or provider obtain the security token from a third party token store service and validate the proof. Inherent with the first solution is that it assumes that the two parties have used some mechanism to establish a trust relationship for use of the security token beforehand. The latter case demands either the requester or the provider to obtain the security token before being able to invoke the Web Service.

Defining policies regarding reliable and secure messaging is the focus of the WS-Policy [4] specification which includes alongside the security token requirements, privacy attributes, encoding formats and supported algorithms.

Since there are already proposals how to align WSMO with WS-Policy [25] it can be expected that Semantic Web Services will ultimately represent such policy assertions as logical conditions. However this is not achieved yet and therefore security constraints can neither be expressed in the goal nor in the service description.

Negotiation

Negotiation templates

The negotiation stage of the E-Commerce interaction lifecycle refines the agreed service specification from the matchmaking phase to a concrete agreement between two parties [47].

From a business point of view, negotiation is a key factor organisations use for achieving cost reduction in E-Commerce [27]. Automated negotiation in current B2B protocol standards is only addressed in a limited way. OASIS, the standardisation body of ebXML is working on requirements for automated negotiation processes to compose/negotiate a Collaboration Protocol Agreement (CPA) from the Collaboration Protocol Profiles (CPPs) of two prospective trading partners [44]. It is not yet defined as a standard and there are no services available offering such kind of negotiation protocol. Only if two partners agree on negotiation protocols based on the same XML template, this process can be automated in the filtering phase. Furthermore without ontologies it can not be guaranteed that the requester and the provider are referring to the same good.

SWS frameworks have to propose negotiation templates in order to be able to match the protocol supported by the service requester with those used by the service provider. Negotiation templates refer to a common ontology accepted by all participants in the negotiation [47]. They define a schema for valid negotiation proposals that participants submit to each other. The result of the negotiation process is an agreement on the terms which have been relaxed to negotiation. In most cases the price based on the quantity, delivery time, warranties etc. will be subject to negotiation.

Delivery Choreography

Standards and mappings for Public Processes

As presented in section 3.3 in SWS frameworks mediator systems can compensate between differences in the clients communication pattern and the Web Services communication pattern in order to obtain equivalent public processes. Although this is possible in theory only if the two public processes are compliant to some standard or at least the concepts describing the states of the process (the business document standards described above) it is possible to achieve the mediation by applying solvable mismatch rules in the service delivery without any human intervention. If two public processes are heterogeneous to a degree where differences can not be mediated by "intelligent" mediators, transition rules have to be defined at design time. They are subsequently used to identify the equivalences between the two processes which are then applied at run time on the particular process instances. Since the partners in a dynamic environment are not known beforehand standardised public process ontologies similar to the B2B protocol initiative proposal have to be included in the SWS frameworks in order to limit the mediators necessary for resolving process heterogeneity.

3.5 Prerequisites of SWS in intra-EAI

We have shown the benefits of SWS in EAI scenarios in section 3.3; now we have to call the application of SWS to intra-EAI into question. One could claim that heterogeneous business logics and data formats are not present within a company? Furthermore some might state that a Service Discovery is not required, since the functionality exposed by different applications within the organisations boundaries is known to system architects at design time?

First one has to consider that an arbitrarily large organisation consists of a series of sub units, which are set up to deal with the various problems the firm needs to solve to continue existing [41]. These sub units (departments, divisions, etc.) require inputs of one kind or another to do their job.

Organisational and behavioural psychologists have looked into the reasons why sub units do not discover services automatically from within rather than without the organisation. Cyert and March [15] developed the concept of a loose and shifting "coalition" that selects organisational goals. Organisational subunits are assumed to be independent and to deal incrementally and disjointedly with one problem and one goal at a time while being subjected to short-run reactions and to short-run feedbacks on their actions. Over time, as sub units repeatedly alter their goals and relationships to

local environments, the organisation becomes transformed. This leaves little room for the discovery of new services in other, possibly unheard, areas of the existing organisation. Hence dynamic discovery within an organisation can be as beneficial as it is between organisations in B2B E-Commerce.

To be able to discover these services in other sub units the same extensions to the traditional SOA (see section 3.2) have to be made.

This implies that we have to examine if the challenges on SWS frameworks to enable a dynamic environment for inter-EAI as described in the previous paragraph apply for intra-EAI as well.

Matchmaking

Non-functional properties

Since every organisation consist of a limited amount of sub units, these units itself are known to each other and as they conduct business on a daily basis, organisational agreements [39] replace Service Level Agreements (SLAs) in the usage of services. When interests of the business-units and the corporation are aligned, sub unit managers will be motivated to pursue synergies with their sister sub units. It has value-creating potential [39], hence it can be assumed that service usage between sub units is agreed on beforehand.

Only properties related to the availability of single services remain an issue in intra-EAI as well. An organisational wide policy on how to conceptualise these properties can help to include them in the Matchmaking. Another possibility is a monitoring of services within an organisation, whereas every sub unit has access to the metrics to have preferences defined in the filtering phase based on this evaluation metrics.

Ontologies for Business Documents

In fact the existence of ontologies to define the business content (the vocabulary) of messages involved in the interaction between two services remain an issue in intra-EAI as well. One possible solution is again an organisation wide policy on how to conceptualise the relevant domains attached to the messages exchanged within the organisation.

As semantically annotated Web Service interface have to be developed in any case, it is sufficient if this development is coordinated on a strategic level throughout the organisation. In this way an agreement can be enforced rather than be achieved by a mutual agreement between different business partners on a standard in vertical markets as it is the case in inter-EAI.

Policies as logical conditions

Defining WS-Security Policies in logical terms to place constraints on the chosen services is usually not an issue within organisations. Communication between sub units is either over secure ethernet or over a Virtual Private Network (VPN) applying for example traditional transport-layer security mechanisms, such as SSL (Secure Sockets Layer) or TLS (Transport Layer Security). Therefore the security, non-repudiation, confidentiality and integrity of message is guaranteed for message transfer between Web Service as it is for every other communication within an organisation's network.

Negotiation

Negotiation templates

Within organisations the filtering of a result set derived in the Matchmaking is based on the functionality of the service and preferences rather than based on negotiation over the price. Sub units will endeavour to share tangible and intangible resources between business-units to lower joint costs of production (Economies of scope) [23]. Price is merely used for cost accounting purposes and negotiations over the price are not undertaken.

This implies for the filtering that either simply the service with the lowest price (if the functionalities of the discovered services match exactly) or the service with the highest degree of match calculated in the Matchmaking phase will be chosen.

Delivery Choreography

Standards and mappings for Public Processes

This is directly related to the ontologies for business documents. Again as long as an agreed set of ontologies is used within a company, only a limited set of mediators is required to mediate between behavioural heterogeneities.

Now as we have shown that dynamic discovery and invocation by applying current SWS frameworks is possible between organisational sub units under certain circumstances, we show how the Web Service Execution Environment (WMSX) can be applied in a concrete use case.

4. WSMX IN EAI

WSMX as a sample implementation for WSMO provides in its first version implementations of all components constituting a SSOA on different levels of maturity.

4.1 WSMX functionality and architecture

WSMX exposes its functionality as a Web Service by four operations [51]:

1. Store WSMO entity

   This entry-point provides requesters and providers to store Web Service descriptions and choreography interface descriptions (public processes) as well as any WSMO-related entity (like ontologies, goal templates etc.) to make them available for public use.

2. One-way goal execution

   The service requester expects WSMX to discover and invoke the Web Service by providing a formal description of a goal (see section 3.1) and a fragment instance of an ontology. This scenario is comparable to a traditional Web Service invocation where the provider and its associated public process is known to the requester beforehand. It remains in the responsibility of the requester to provide permanent addressing to pick up possible results by WSMX.

3. Web Service Discovery

   A single Web Service based on the selection within WSMX or a list of Web Services is provided when invoking this operation for a given goal and an instance of an ontology. Each Web Service description returned back is already mediated to the requester ontology if necessary. The required exchange message patterns to invoke the Web Service (its public process) is also provided in the result set.

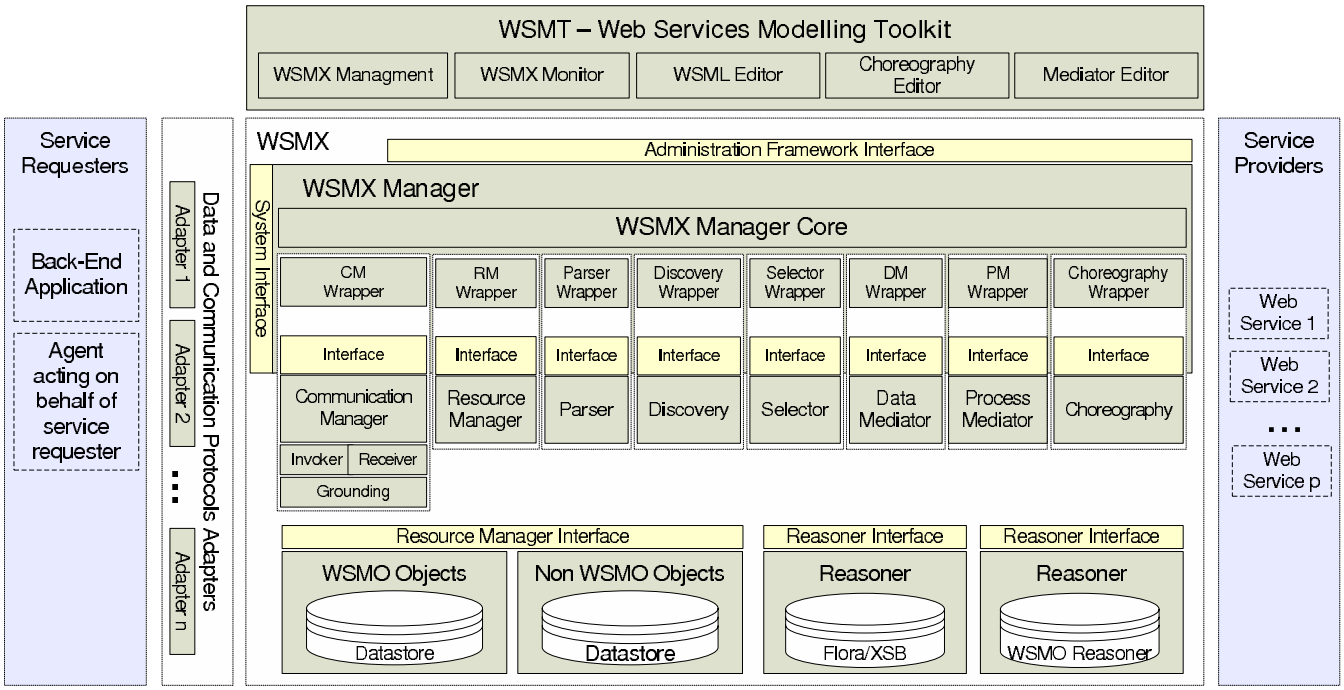

Figure 6: Simplified WSMX Architecture cf. [47] for the complete picture

4. Two-way goal execution

   Once the service requester knows what Web Service to use, this operation is used to invoke the Web Service. The advantage over a traditional Web Service invocation is that a back-and-forth conversation can be carried out by WSMX to provide all the necessary data to make the execution of this Web Service feasible. For that the choreography of the requesting Web Service (its public process) has to be known to WSMX, either provided within the message or registered before through the "Store WSMO entity" operation.

What follows is a brief description of the WSMX components relevant in this use case (see Figure 6). A detailed explanation of all components can be found at [50].

- **Message Adapters:** As long as back end applications do not natively support a WSML compliant Web Service Interface adapters are required to transform the format of messages or even extracted data from an API into the WSML compliant format understood by WSMX.
- **Resource Manager:** This component is responsible for managing the repositories to store definitions of any WSMO and non-WSMO related object in WSMX. The service registry can either be centralised or decentralised, whereas the current architecture only deals with a decentralised repository in each WSMX.
- **Discovery:** The Discovery component attempts to match the service requester's goal to a capability stored in any known repository. The functionality corresponds to the dynamic discovery described in section 3.2.
- **Selector:** It provides a dynamic selection of the discovered Web Services in the Matchmaking process. The selection is currently based on a limited set of non-functional properties.
- **Data Mediator:** The task of this component is to transform the incoming data from the terms of the requester's conceptualisation (source ontology) in terms of the providers's conceptualisation (target ontology). Data mediation is based on paradigms of ontology management, ontology mapping and aligning in particular.
- **Process Mediator:** The role of the Process Mediator is to put together the necessary means for the runtime analyses of two given choreography instances and to compensate the possible mismatches that may appear (see section 3.3).
- **Communication Manager:** It has two major tasks: first, to handle the various invocations that may come from requesters and second, to invoke Web Services and to retrieve the results of these invocations back to WSMX. Currently, even if a semantic description is provided for a certain Web Service capability, the actual invocation still has to be made in a classical way, by representing all the data needed for the invocation in the required XML format. In order to make the bridge between the semantic descriptions used within WSMX and the classical syntactic Web Service descriptions, WSMX provides the necessary means for lifting non-semantic descriptions (e.g. XML messages and XML schemas) to a semantic level and to lower the elements semantically described (e.g. ontology instances and concepts) to the level required by the classical approaches.

4.2 Use Case - WSMX in intra-EAI

First we define some prerequisites on the architecture itself in order to depict a high-level intra-EAI use case. To keep the use case as general as possible, we do not assume WSMX to be installed either on the requester or the provider side. Since WSMX exposes all its functionality itself as a Web Service, it acts in the use case as any other component exposed as a Web Service. The only difference is that the requesting agent has to be bound to the WSMX

Web Service at design time, since it performs the discovery of SWS and therefore can not discover itself.

However, it is of no relevance where the WSMX systems physically resides. Since it is more likely that one sub unit publishes the service description of functionality they would like to expose for reuse in their own WSMX repository, we can expect several WSMX systems installed in the organisation. This would not hinder the discovery of service descriptions stored in different WSMX systems as long as they are known to each other. Therefore we assume that the service providers have already registered their Web Service in at least one internally operated WSMX repository. Further the service requesters also have their public process registered with any WSMX. If they are stored in two different systems we assume that the two WSMX repositories are known to each other and a physical connection is available.

The following use case describes the execution semantics of WSMX in a concrete scenario and shows how sub units are able to dynamically discover and invoke services within the organization's boundaries. The operational flow of the discover and invoice operation are depicted in figure 7.

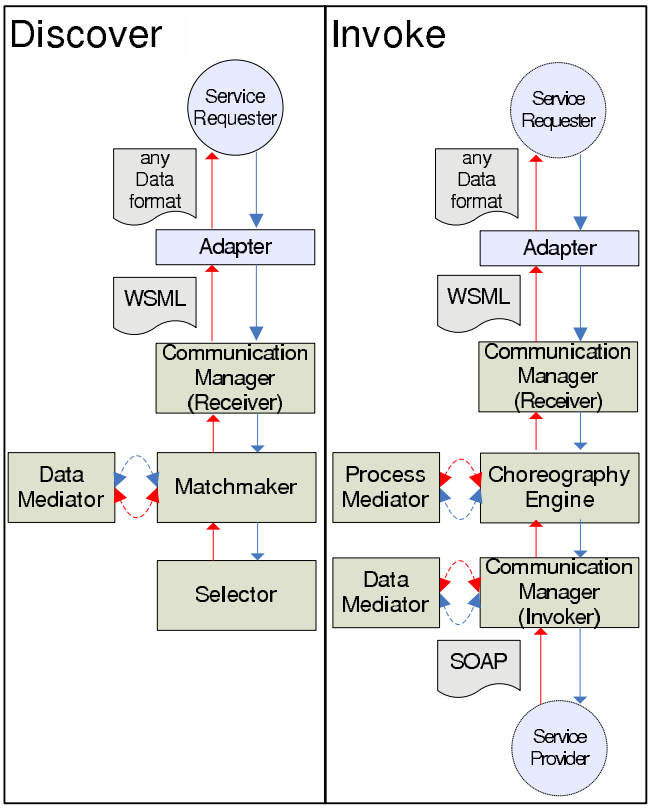

Figure 7: Simplified Operational Aspects of WSMX

Let us assume a stock purchase by an investment fund department within a multinational operating bank. This division, which we will further call HFD (hedge fund division), manages a global hedge fund. Within its business it has to buy all kind of certificates from different markets. Typically purchases of certificates on different markets are managed by so called settlement divisions. In large-scale banks there might not be only one settlement, but several ones in different countries with overlapping functionality. Similarly different funds are managed in different departments, either by outsourced sub companies or independent entities within the bank. The HFD now wants to buy one million shares of the Bridgestone Co., listed on the NIKKEI index and traded at the Tokyo Stock Exchange (TSE), by using settlement services available somewhere within the bank. The proprietary application managing the portfolio of the HFD was configured in a way that it automatically orders one million shares of the Bridgestone Co. if the share price drops below 12 Yen.

Leveraging a SSOA the application (service requester) sends the goal to buy one million Bridgestone shares at the current market price at the TSE to the WSMX Discover operation (see section 4). When invoking this operation the application defines in the preferences attribute if a set of Web Services or a selected Web Service based on non-functional properties is desired. If this application does not support a Web Service interface, adapters have to provide the connectivity in the SSOA.

WSMX tries to find matching Web Service capabilities based on different levels of logical reasoning (see section 3.2). If WSMX is not used on a global level within the organisation and the two sub units use different WSMX systems the actual result set will also contain service descriptions stored in the WSMX repository used by the settlement division to publish their service description.

If the HFD and the settlement division use dissimilar ontologies to describe their goal and their service description the mediator component can compensate the semantic mismatch of the two descriptions in the Discovery phase. As we mentioned above we assume that a limited set of ontologies is used within the organisation and the necessary mediators are in place.

If the result of the Matchmaking was a set of Web Services and a WSMX internal selection is not desired, the requesting application has to select the actual Web Service to invoke. As we mentioned above in intra-EAI this selection can in most cases be based on the degree of functionality match since non-functional properties like the price are not an issue.

Now the requesting application calls the two-way goal execution operation of WSMX to finally invoke the selected Web Service. Included in the message sent to the WSMX server is the actual ontology instance data (e.g. the share name, the amount of shares etc.) and the choreography interface (public process) or a URI reference to its stored definition. In the most likely case that the requester's and provider's communication patterns are heterogeneous the process mediator would resolve the mismatch and control the back-and-forth of messages between the requester, WSMX and the provider. Again we assume a limited set of ontologies with the respective mediators available to allow a fully dynamic resolution of process heterogeneity. The result of the invocation is the final state in the communication pattern sent back by WSMX to the requester. In our case it would probably be some acknowledgment when the shares were bought and to which account they were credited to.

5. RELATED WORK

Since our architecture applies WSMO as the underlying SWS framework, related work is separated into other SWS frameworks and EAI systems based on the SOA paradigm.

Beginning with the latter (Web Service enabled SOAs) we have to identify a number of commercial applications. Among these systems are IBM WebSphere MQ[10], IONA Artix [11], webMethods Fabric [12], BEA WebLogic Enterprise Platform [13] and others. All these products have in common that they do not facilitate semantic annotation of the published services. They either rely on UDDI

[10]see http://www-306.ibm.com/software/integration/wmq/
[11]see http://www.iona.com/products/artix/
[12]see http://www.webmethods.com/meta/default/folder/0000006124
[13]see http://www.bea.com/

for service registry or they utilise CORBA Trading Service [33]. Latter complements the meta-database which is used by an Object Request Broker (ORB) and the Naming Service. Again the discovery service acts like yellow pages with merely syntactical search functionalities.

Beside WSMO there are two other frameworks providing the semantic mark-up for automated service discovery; OWL-S (formerly DAML-S) [46] and METEOR-S [38] and a third research project called IRS3 [17] working on a SWS execution environment.

OWL-S [46] is an OWL-based Web Service ontology. For OWL-S only separate tools are available such as a composer, a matchmaker and an editor. A complete toolset for design- and runtime is not yet available for download. A dynamic binding of Web Services by augmenting the BPWS4J [14] implementation of BPEL with a semantic discovery service is described in [31]. Since the BPEL specification itself restricts the description of service partners to syntactic WSDL 'portType' definitions, this solution builds upon a Web Service that can discover service partners at runtime based on their OWL-S descriptions. This discovery service is then bound to a BPWS4J process at design time.

METEOR-S [49] works with existing Web Services technologies and combines them with ideas from the Semantic Web to enable Web Service discovery and composition. In contrast to WSMO and OWL-S it does not introduce a new ontology language, but currently uses DAML+OIL and RDF(S) to map WSDL message types (inputs, outputs) and operations to concepts in domain ontologies. The main components of METEOR-S are an Abstract Process Designer, a Semantic Publication and a Discovery Engine, a Constraint Analyzer, and an Execution Environment. The Abstract Process Designer allows a user to create the process model by using BPEL constructs. The Semantic Publication and Discovery Engine provides semantic matching based on subsumption and property matching. The Constraint Analyzer dynamically selects services from candidate services which are returned by the discovery engine. The Execution Environment performs the binding of services returned by the Constraint Analyzer and converts abstract BPEL to executable BPEL.

A similar tool to WSMX is IRS3 [17], a Web Service execution environment compliant to WSMO. With the current version of IRS3 a service provider can create a WSMO service description that can be published against their service on the IRS3 server. Once the service description is available, a goal can be described in WSMO and bound to the published Web Service description using a mediator. This binding is still at design time, but the use of mediators to link goals and services removes the manual hard-wiring required for standard Web Services.

6. CONCLUSION

In this paper we have proposed to extend the notion of Service-Oriented Architectures by Semantic Web Services. We applied the principles of the Web Service Modeling Ontology (WSMO) to this new type of architecture (SSOA) and showed how EAI benefits by it. WSMO as a formal ontology for describing various aspects related to Semantic Web Services applied in a SSOA eventually allow dynamic discovery and invocation of services published in the architecture.

Furthermore with interfaces formally defined in WSML integration for developers become a good deal easier. We have shown how the functionality in the transformation and process layer is greatly improved. Instead of mapping between XML Schemas as it is done in traditional SOA, in our architecture mediation takes place at a higher level of ontologies, which greatly reduces the required mappings. By applying SWS principles in the process layer, mediator systems can dynamically compensate the clients or the Web Services communication pattern in order to obtain equivalent public processes.

Since the automated discovery and execution of Web services is not facilitated by a conceptual model alone we have identified some challenges on SWS frameworks and standardisation efforts to be further addressed to ultimately reach the goal of dynamic environments. These issues included to define standards for business document ontologies, ontologies for expressing Service Level Agreements, business rules and business goals, negotiation protocols and finally policy assertions as logical conditions (security, reliability etc.).

We concluded that dynamic discovery and invocation is already possible in intra-EAI under certain assumptions. To showcase the potentials of a SSOA we applied WSMX, as a first representative of a SSOA, to an internal integration use case. Finally we described how WSMX addresses the activities we defined in a lifecycle model for a SSOA.

The authors are part of the WSMO and WSMX[15] working group and will mainly contribute to the further development of the choreography and orchestration component. The challenges mentioned in this document are not unknown to the WSMO working group and are addressed in different initiatives. Currently work begun on extending non-functional properties. As already mentioned another initiative is to ontologize EDI [19] and work on policies is on its way.

7. REFERENCES

[1] T. Andrews et al. Business Process Execution Language for Web Services, Version 1.1. `ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf`, 2003.

[2] A. Arkin et al. Web Service Choreography Interface (WSCI) 1.0. W3C Note, `http://www.w3.org/TR/wsci/`, 2002.

[3] B. Atkinson et al. WS-Security. `http://www-106.ibm.com/developerworks/webservices/library/ws-secure/`, 2002.

[4] S. Bajaj et al. Web Services Policy Framework (WS-Policy). `http://www-128.ibm.com/developerworks/library/specification/ws-polfram/`, 2004.

[5] C. Bussler. B2B Protocal Standards and their Role in Semantic B2B Integration Engines. *IEEE Data Engineering Bulletin*, 24(1):3–11, 2001.

[6] C. Bussler. The Role of B2B Protocols in Inter-Enterprise Process Execution. In *Proc. of Technologies for E-Services (TES)*, 2001.

[7] C. Bussler. *B2B Integration*. Springer-Verlag, 2003.

[8] C. Bussler. The Role of Semantic Web Technology in Enterprise Application Integration. *IEEE Data Engineering Bulletin*, 26(4):62–68, 2003.

[9] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`, 2001.

[10] J. Clark et al. ebXML Business Process Specification Schema 5, Version 1.01, 2001.

[11] L. Clement, A. Hately, C. v. Riegen, T. Rogers, et al. UDDI Version 3.0.2. `http://uddi.org/pubs/uddi_v3.htm`, 2004.

[14] see http://www.alphaworks.ibm.com/tech/bpws4j

[15] WSMX working group - http://www.wsmx.org

[12] S. Colucci, T. D. Noia, E. D. Sciascio, F. Donini, M. Mongiello, G. Piscitelli, and G. Rossi. An Agency for Semantic-Based Automatic Discovery of Web-Services. In *Proc. of Artificial Intelligence Applications and Innovations (AIAI)*, 2004.

[13] S. Craggs. Raising EAI Standards. `http://www.eaiindustry.org/docs/Raising%20EAI%20standards.pdf`, 2003.

[14] F. A. Cummins. *Enterprise Integration: An Architecture for Enterprise Application and Systems Integration*. Wiley, 2002.

[15] R. M. Cyert and J. G. March. *A behavioral theory of the firm*. Prentice-Hall, Englewood Cliffs, NJ, 1963.

[16] J. d. Bruijn. The WSML Specification. WSML Working Draft, `http://www.wsmo.org/TR/d16/`, 2005.

[17] J. Domingue, L. Cabral, F. Hakimpour, D. Sell, and E. Motta. IRS-III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services. In *Proc. of the Workshop on WSMO Implementations (WIW 2004)*, 2004.

[18] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2), 2002.

[19] D. Foxvog and C. Bussler. Ontologizing EDI: First Steps and Experiences. In *to appear in Proc. of the International Workshop on Data Engineering Issues in E-Commerce (DEEC)*, 2005.

[20] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen. Soap version 1.2. `http://www.w3.org/TR/soap12`, 2000.

[21] Integration Consortium. Thoughts from the EAI Consortium Leaders: Avoiding EAI Disasters. `http://www.dmreview.com/editorial/dmreview/print\_action.cfm?articleId=8086`, 2004.

[22] S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation*. International Thomson Computer Press, 1996.

[23] R. W. J.C. Panzar. Economies of scope. *American Economic Review*, 71(2):268–272, 1981.

[24] M. Keen et al. *Patterns: Implementing an SOA Using an Enterprise Service Bus*. IBM Redbooks, 2004.

[25] J. Kopecký and D. Roman. Aligning WSMO and WSMX with existing Web Services specifications. WSMO Working Draft D24.1 v0.1, `http://www.wsmo.org/2005/d24/d24.1/v0.1/20050117/`, 2005.

[26] H. Kreger. Web Services Conceptual Architecture (WSCA 1.0). `http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf`, 2001.

[27] M. Kumar and S. I. Feldman. Business Negotiations on the Internet. In *Proc. of the Internet Society (INET)*, 1998.

[28] D. D. Lamanna, J. Skene, and W. Emmerich. SLAng: A Language for Defining Service Level Agreements. In *Proc. of the Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, 2003.

[29] A. Laroia and L. Sayavedra. EAI Business Drivers. *eAI Journal*, (2):27–29, 2003.

[30] L. Li and I. Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. In *Proc. of the International Conf. on the World Wide Web*, 2003.

[31] D. J. Mandell and S. A. McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. In *Proc. of the International Semantic Web Conference (ISWC)*, pages 227–241, 2003.

[32] S. McIlraith, T. Son, and H. Zeng. Semantic Web Services. *In IEEE Intelligent Systems. Special Issue on the Semantic Web.*, 16(2):46–53, 2001.

[33] B. Medjahed, B. Benattalah, A. Bouguettaya, A. Ngu, and A. Elmagarmid. Business-to-business interactions: Issues and enabling technologies. *The VLDB Journal*, 12(1):59–85, 2003.

[34] A. Mocan and E. Cimpian. Mediation. WSMX Working Draft D13.3 v0.1, `http://www.wsmo.org/2004/d13/d13.3/v0.1/20040906/`, 2004.

[35] E. Newcomer and G. Lomow. *Understanding SOA with Web Services*. International Thomson Computer Press, 2004.

[36] J. O'Sullivan, D. Edmond, and A. H. M. ter Hofstede. Formal description of non-functional service properties. Technical report, CITI, 2005.

[37] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic Matching of Web Service Capabilities. In *Proc. of the International Semantic Web Conference (ISWC)*, 2002.

[38] A. Patil, S. Oundhakar, A. Sheth, and K. Verma. Semantic Web Services: Meteor-S Web Service Annotation Framework. In *Proc. of the International Conference on World Wide Web*, pages 553 –562, 2004.

[39] C. Perrow. *Complex organizations: a critical essay*. McGraw-Hill, 1986.

[40] C. Preist. A Conceptual Architecture for Semantic Web Services. In *Proc. of the International Semantic Web Conference (ISWC)*, 2004.

[41] S. W. R.H. Coase, O. Williamson. *The Nature of the Firm*. Economica, University of Chicago Press, 1937/1991.

[42] D. Roman, H. Lausen, U. Keller, et al. Web Service Modeling Ontology (WSMO). WSMO Final Draft, `http://www.wsmo.org/2004/d2/v1.0/`, 2004.

[43] W. A. Ruh, F. X. Maginnis, and W. J. Brown. *Enterprise Application Integration: A Wiley Tech Brief*. Wiley, 2000.

[44] M. Sachs et al. Negotiation Requirements. `http://www.oasis-open.org/committees/download.php/1577/Negotiation.req.06Mar02.pdf`, 2002.

[45] D. Skeen. Business Vocabulary Management. *Business Integration Journal*, (7):10–12, 2003.

[46] The OWL Services Coalition. OWL-S: Semantic Markup for Web Services. `http://www.daml.org/services/owl-s/1.0/owl-s.pdf`, 2004.

[47] D. Trastour, C. Bartolini, and C. Preist. Semantic Web Support for the Business-to-Business E-Commerce Pre-Contractual Lifecycle. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 42(5):661–673, 2003.

[48] U. Keller, R. Lara, A. Polleres. WSMO Discovery Engine. WSMO Working Draft, `http://www.wsmo.org/2004/d5/d5.1/v0.1/20041112/`, 2004.

[49] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Journal of Information Technology and Management*, 6(1):17–39, 2005.

[50] M. Zaremba and M. Moran. WSMX Architecture. WSMX Working Draft D13.4, `http://www.wsmo.org/2005/d13/d13.4/`, 2005.

[51] M. Zaremba and E. Oren. WSMX Execution Semantics. WSMX Working Draft D13.2 v0.2, `http://www.wsmo.org/2005/d13/d13.2/v0.2/20050202/`, 2005.