Execution of Semantic Services in Enterprise Application Integration

PETER MARTINEK1, BALAZS TOTHFALUSSY2, BELA SZIKORA1
Department of Electronics Technology1
Department of Automation and Applied Informatics2
Budapest University of Technology and Economics
Goldman Gy. tér 3., 1111 Budapest
HUNGARY

Abstract: - In this paper, we present an approach for the execution of semantically enriched services in Enterprise Application Integration (EAI). After the brief overview of current researches and literature the basic definitions and concepts are revealed. Our integration platform is based on a typical Service Oriented Architecture (SOA) solution, which consists of a service registry, a process designer and a run-time engine. Furthermore there are some additional components for realizing semantic enrichment e.g. the semantic profiler and the Ontology. The focus of the paper is the preparation for the process run-time. We propose a mediator based approach where two data transformations are assigned to each service during a so called annotation process. The standard services of ERP, CRM, SCM etc. systems are encapsulated into these mediator services which makes possible to apply them in our semantic integration framework. Furthermore our method enables that our semantically composed processes and their services remain executable on current process run-time standards.

Key-Words: - Enterprise application interoperability, Semantic services’ run-time, Collaborative business processes

1 Introduction

Organizations are hard to imagine without complex software systems today. Information systems support the everyday working processes of companies, non-profit organizations and governmental organizations. Spreading of such systems indicated the need for exchanging data and realizing communication between them. Thus companies mainly focus on business collaboration [1] instead of applying new enterprise systems today. Collaboration is needed both in intra- and inter-organizational level, accordingly we can talk about intra- and inter-organizational business collaboration [2].

A lot of current approaches in enterprise application integration (EAI) build upon a Service Oriented Architecture (SOA) [3]. The main concepts and definitions of SOA are briefly described in the next paragraph.

The basic building construct of SOA are services. There are many definitions for services in the literature. In our case let say, that invoking a service means the execution of a function of an enterprise system. So, services represent exposed functionality of enterprise systems. There are service requesters and service providers in a specific SOA integration scenario. Requester invokes services offered by the provider which means the execution of a function, e.g. a query, a calculation or a simple data exchange.

Integration scenarios are solved by composed processes. The processes consist of services, and rules connecting them, for example in the simplest case the process is only simple chain of services. This may result that participating enterprise systems are both service requesters and providers in a specific integration scenario. Besides building processes out of services can solve integration problems it is also able to create new capabilities on existing services of applications.

Evolution of the semantic web raised new possibilities also in EAI. Although the typical SOA technologies rely only on syntactical approach for process based integration [4], adopting the fundamentals of the semantic web into the world of EAI is reasonable. For example it is also important to know about the capabilities of participating services while designing a collaborative business process. Classification of available services due to a common, global schema describing the concerned business area may also be necessary.

To be SOA enabled, software vendors of enterprise application systems already created services to their systems. However these are mainly described on a rather technological way and can be understood only by the experts of the specific system. Attaching some semantic meta-information to the services may help business consultants by designing collaborative business processes.

There are numerous current proposals which offer full approach e.g. methodologies and working tools for supporting the semantic extension of services. In most of them the services are stored by semantic repositories.
The repositories assign various indexes to the services, which makes possible to compose collaborative processes easily by finding (discovering) services by their semantic matter. However from the point of view of business logic we can create a perfect process now, our newly composed process won’t be able to run and perform the operations what it was designed for. The descriptions of the services were extended by semantic content, but the interface of services (required input and output data format and communication protocol) of standard services remained intact. Thus there is a semantic gap between the description of input and output data schema of the service and its semantic description. Defining transformations to overlap this semantic gap between the intact services of standard systems and the applied semantically described services (semantic services) is a good start to solve the problem but a framework for executing the transformations itself is also required.

The rest of the paper is organized as follows: Section 2 compares our approach with related work. Section 3 briefly describes our methodology for creating semantic services. Section 4 provides for in detail description of the transformation creation process and our mediation technique for the process run-time, and section 5 draws conclusions and outlines future work.

2 Related work
There are numerous researches in literature about interoperable systems in service oriented architecture. For example, see [5, 6, 7, 8, 9]. There are many tools and solutions to design and run standard BPEL processes, for example the Oracle Fusion Middleware [10] or the IBM Websphere [11]. But usually they don’t provide the extension to characterize the services also with semantics. Without information about service capabilities and behavior it is hard to compose collaborative business processes but adding semantics already excludes the pure using of such standard run-time environments.

The Intelligent Software Agents Lab [12] at Carnegie Mellon deals with an SOA approach which is using OWL-S service description for choreography. They also developed tools for supporting the creation of the OWL-S description from a WSDL [13] basis and publishing them into a UDDI [14] registry. They, however, use the OWL-S to describe the processes as well, while we use BPEL for this purpose. In the last 2 years BPEL became the most popular standard in business process description and run-time. We think, that it is not only more reliable, but more scalable than OWL-S. By applying WSDL-S for describing the services it was also possible to add semantic extensions and preserve services in a BPEL compatible format at the same time. Grossmann et al. [8] present a behavior based integration methodology for business processes. By using integration operators, the authors can create, deal and finalize compositions between them. However the approach in this work is based only on the observed states of the processes and the behavior of participating services. This may lead to a valid transformation of processes but any integration is hardly realizable without taking into consideration the differences in the input and output data schemas of services and processes. Indeed, processes coming from different companies or application systems may use different semantic conceptualization to describe the same real world concept. This raises several issues which must be taken into consideration as well if we are planning to go also for run-time.

3 Semantic Web services
The vendors of enterprise application systems have created standard services in their systems to prepare for SOA based integration scenarios. Unfortunately, the offered services are usually described only by technical information. This information is enough to invoke the service with some test data, but not enough to understand its behavior. Business process composition also requires attached pre- and post- conditions, which should be taken into consideration by invoking the services. Our architecture offers tools for enterprise application developers to attach such information to service descriptions. Furthermore, it is possible to label a service operation upon the identified functionality of services. A business ontology serves as a common reference to the labeling [15]. Fig. 1. shows our simplified system architecture.

Using the semantic profiler tool one can attach additional information about service behavior to the service descriptions. The semantically enriched descriptions are stored in the semantic registry. Using the discovery interface of the registry we can already identify existing enterprise application services and compose process from them. However the composed process won’t be ready to ‘go live’. Detailed information about the schema of input and output data of the services is also necessary to the run-time [4]. Real world concepts are described in the databases of enterprise applications. It is obvious, that the identifiers (e.g. name, ID, attributes) of the same real concepts may differ in the applications developed by different vendors. So services applied in composed processes rely on their own schema in their input and output data. To create mappings between them, semantic relationships between their concepts must be known. Only concepts representing the same real world entities can be mapped to each other. For example, if a service requires the name
Creation of the mapping of each service input to each service output in an integration scenario requires enormous resources. Furthermore, this structure is also hard to maintain. If a new service is added to the integration scenario, the mapping to every other service has to be defined [16].

Defining a global schema reduces the complexity of the system. This global schema covers all possible real world concepts to the specific scenario. The services are mapped only to the global schema concepts, and the communication between services in processes is done on the level of global schema concepts. After that the mapping of services on the process level is not a complex issue any more because same real world concepts are represented by the same concepts in the global schema. Our global schema concepts and the taxonomy are also stored in the Ontology [15].

The first step of enabling services of standard systems to participate in our semantic integration scenario is the semantic service annotation. In this, the service description is extended with semantic meta-information. Actually these are references to the semantic concepts stored in the Ontology. (The semantic profiler supports the identification and association of appropriate concepts e.g. service capabilities, input/output data types, service’s pre- and poststates. Easy to use graphical interfaces were also implemented for the business consultants.)

The semantic gap between the global schema and the services’ so called local schema should be bridged afterwards. Two types of transformations are defined to each annotated service:

- The down-cast transformation is used to connect global schema data input of the semantic service and services local schema input, and
- The up-cast transformation is used to transform the service reply from the local schema concepts into valid concepts of our global schema.

Transformations are stored in a transformation repository and are referred also from the semantically extended service description.

The last step of the semantic service creation is the creation of a directly invokable interface. This is done by the encapsulation of the annotated service which is actually a web service generation in technically sense.

The next chapter presents our approach, methods and tools for the transformation creation and service encapsulation.

4 Mediator services and transformations

In this section we concentrate on the solutions for the problems and concepts e.g. semantic gap between standard and semantic services, transformation creation, invokable services generation, introduced in the previous section. As it was demonstrated, there are two layers of services: the layer of standard services (native services), which hold some functionality of an enterprise system provide value for us and thus it must be integrated and the layer of encapsulated services (mediator service), which communicates towards the outer world based on the common (global) schema concepts. First, we summarize the expectations that the implemented mediator service should meet:

1. Provides an endpoint for service calls, which is a well-known XML web service interface, described by a pure WSDL file.
2. Catches the incoming requests and applies down-casting based on the invoked operation.
3. Relays the invoke containing the transformed information to the native service.
4. Catches the reply of the native service and applies upcasting transformation (actual operation name must be determined).
5. Replies the transformed information to the client (service invoker) of the mediator service.

The current standard for services are Web services and the description of them is done by WSDL. We rely on SA-WSDL descriptions, which are the semantically enriched versions of pure WSDL documents. An SA-
WSDL document carries all the required information for creating mediator services: modelReference extension attributes on certain elements of the WSDL document, which connects the simple syntactic data types with the ontology level concepts, and upcasting and downcasting information for the web service operations. This means that the description can be enriched with semantics but the service itself (and its standard interfaces) remains intact. Although this section concentrates on mediator services, where the “data related” semantic information is used, please note, that modelReference attributes contain also other semantic information than this of the data level. State annotations, Service taxonomies and other descriptions of service behavior must also be recognized, but are to be ignored from the point of view of the mediation.

Our service annotation methodology [16] defines the exact set of WSDL elements, where the data related annotations must appear, so the SA-WSDL document is parsed and used based on well-defined rules. As an example we specify the place of upcasting and downcasting information later on. For input, it must be on the topmost XSD element (to which the WDSL input element refers), which holds all possible input arguments. This is needed, because the argument number of the native and mediator services may not be the same. If multiple modelReference values are used, then the mediated service will have more arguments than the native one. In this case, the downcasting transformation must join these arguments into the native version. This method is the same for outputs. Furthermore for the output elements our approach adopted the concept of conditional outputs from the OWL-S specification. Conditional outputs are the list of possible output, but it is not guaranteed, that all output elements is contained by the response. A simple example can be a book search service, which returns the book record if it is found, or a simple string message (“Book can not be found”), if it is not contained by the database. If conditional outputs are present, than on the topmost element of the choice (possible output) must refer to the upcasting transformation. This transformation may split the native argument into more ontology level arguments or simply map them.

As defined above, the mediator service has to implement five steps to accomplish its goal. Functionality of web services is stored in the implementation class, thus the implementation class relays the service request to the native service, and sends the native level data objects to the interface of the implementation class which is also the native interface at the same time. Then where is the transformation done? The answer is a little bit based on the applied technology: JAX-WS 2.0 is used for implementing mediator services and this technology offers so-called message handlers (in previous releases, it was called message interceptors). Message handlers are designed to operate on SOAP messages (SOAP messages are the standard communication blocks of web services i.e. service requests and replies), process or alter them if needed. This is also the case for mediator services, as the transformations must be applied here. In our example, XSLT documents (the documents containing the up- and downcast transformation rules) define the transformation procedure and a logical message handler is introduced, which operates on the SOAP body – this ensures, that the service call preserves all the SOAP header elements, which can contain additional information for the native service (for security, transaction etc). As the message handler alters the SOAP message to set it to the native data types, the

![Fig. 2 Mediator service request-response sequence diagram](image-url)
implementation class can retrieve the information in the form of native data objects. The implementation class only relays the call to the native service so far, but this can be a possible extension point for middleware services as well. After the response is got back, the message handler starts to work again. In this case the uplifting transformations are applied, at the end the returned SOAP message conforms the ontology-level data structure for the client of the mediator service.

The complete service invoke-reply process of the mediator service is shown in Fig. 2.

4.1 Generation of mediator services

Another issue is the creation of the mediator service itself. The developed method is able to perform it automatically, so a code generation step was also implemented. Although this may looks quite simple for the first time, there are more complex issues during the mediator service generation as described by the implementation class and the message handlers in the previous section.

As already mentioned, the mediator service uses the ontology level concepts, which are an abstraction over the available native services data layer (from our point of view). The ontology concepts are described in some ontology language, in our case in OWL. The mediator service must have a WSDL interface, in which the data types must be declared in the form of standard XSD elements, which means, that the ontology concepts must be mapped to XSD constructs. Fortunately, this process can also be automated:

The input of our mapping algorithm is a defined OWL class, which represents an argument of the service request. The algorithm is based on the following rules:

- All DatatypeProperty and ObjectProperty are taken into consideration, which have the upper class as their domain,
- The OWL class is mapped to an XSD complexType (this will be a Java class in the implementation class),
- Every DatatypeProperty is mapped as a “primitive” or “simple” property of the complexType,
- Every ObjectProperty is mapped as a property with a reference to the complexType of the range of this ObjectProperty,
- Recursively the range OWL class of the ObjectProperty is mapped the same way,
- All the mapped XSD complexTypes are merged at the end, and this definition is included in the description of the mediator service.

During the mediator service generation, some technical aspects must be taken into account as well. As a Java-based web service, the whole service must be included in a web archive file, and deployment descriptors should be generated containing the web archive and the web service, etc. This is done by a well defined structure, so a template is used for every artifact of that kind. Freemarker [16] was chosen for defining the templates and generate the instance documents based on the templates, which is very similar to the well-known JSP idea and syntax. During the automated service generation, the following steps are performed:

1. Generate the native service client java classes.

   Because the mediator service is a the client of the native service, with the stub classes, it can send the request quite easily to the native one,

2. Extract ontology references from the semantically enriched service descriptor and convert ontology level classes into XSD definitions,

3. Generate implementation class, service message handler and WSDL definition for the mediator service,

4. Create web.xml and sun-jaxws.xml deployment descriptors,

5. Compile classes and run Java annotation processing tool (apt) to generate JAX-WS artifacts for the mediator service,

6. Package and deploy services (a simple JAX-WS enabled web container is enough to run the mediator service).

4.2 Defining transformations

The role of transformations is described in the chapters above. As already mentioned transformations have the task to bridge the differences between the ontology level and native level XSD structures. As XSD definitions can be visualized by a tree, and properties and nested complexTypes build up a tree quite straightforward, we represent the transformations as converting a tree into another. There are a lot of tools, which can support schema to schema transformation, but at this stage we defined some simple algorithms of our own. Fig. 3 shows our representation for a sample transformation.

As you can see, transformation operations can be defined between tree nodes, where the operations have only one output, and (may) have multiple inputs. For a simple case e.g. a string concatenation, let’s imagine that the
ontology level defines a firstName and lastName property, but the native level requires only a name property. In this case, the firstName and lastName must be concatenated. Another simple example can be that a simple prefix “has” is used as a prefix for concept names on the ontology level. So hasName, hasAddress, hasType are the names for ontology level concepts. It is possible, that the native level only uses name, address and type, respectively. So the value of the element must not be changed, but the tag names change, which is a rename transformation operation.

Transformation operations are grouped based on the target element for which they were defined. Accordingly, the transformations themselves compose a tree, where they are assigned to the nodes of the target tree (in this case, the native level XSD). A template-based engine (Freemarker) was used to generate the XSLT file, which describes the transformation for the mediator service. The transformation usually copies everything (all nodes and attributes) from the inbounding request, and at certain nodes, where the transformation is defined, injects the “transformation code”, into the parametrized XSLT instruction, which conforms to the actual transformation operation. (This method can also be used for the uplifting transformations of course.) After all, this transformation must be referenced from the semantic description of the service.

6. Evaluation of results

To evaluate our results a demo example was also implemented, see figure 4. Simulating services of an SCM and CRM systems we have developed demo services in JAVA, which offered standard Web service interface to the co-operation (providing functions A, B and C). The services providing function B and C were analyzed and encapsulated with our method and a composite process was implemented over their new, semantic interface.

The performance was analyzed using a test machine containing an 1.6 MHz Intel Centrino processor, 2GB memory and running JAVA version 1.5.0.5. The time needed for the mediated service generation, was around 20 sec with the load of the ontology from a web URL, the service generation itself is 7-8 secs and the build (running Ant) is another 4-5 secs. But these have to be done only once for one version during the design time. Running of services does not take significant more time than it takes to run a standard Web service. The time of the whole process execution is comparable to the time of running a standard BPLE process.

![fig4](image-url)
5 Conclusion

The paper has presented an approach for creating transformations and generation of invokable semantic services for a SOA based integration scenario. The developed tools are able to support the IT expert by this process and makes possible to deal with already existing services of standard enterprise systems in a semantically enriched environment. This results an efficient method for realizing business collaboration via enterprise application integration.

Although our method supports the easy creation of transformations, it gives no promotion to this area. Semantic relationships between the elements of complex types will also be evaluated in future work. This can result some proposals for creating specific transformation rules. Furthermore we probably can automatically generate proper transformations in specific circumstances upon that information.

Our method relies at several points on the contribution of IT-experts. IT experts are responsible for the identification of relations between the native schema concepts and ontology level concepts, the creation of transformation defining by several transformation rules and the building of composed processes or process templates in a semi-automatic process composition scenario. This human factor carries the possibility of design errors which are signaled by our tools of course. However some self-adaptation abilities could help to overcome on smaller design errors of the data-annotations and transformations at run-time and some self-healing property of running process instances could result recovering from inconsistent process-states automatically. These can be some major issues of our future work.

Acknowledgement

This paper was supported by the FUSION FP6-027385 Project. We also wish to acknowledge our gratitude and appreciation to all the FUSION project partners for their contribution during the development of various ideas and concepts presented in this paper.

References: