Implementation of Semantic Services in Enterprise Application Integration

PETER MARTINEK[1], BALAZS TOTHFALUSSY[2], BELA SZIKORA[1]
Department of Electronics Technology[1]
Department of Automation and Applied Informatics[2]
Budapest University of Technology and Economics
Goldman Gy. tér 3., 1111 Budapest
HUNGARY
martinek@ett.bme.hu http://www.ett.bme.hu

Abstract: - In this paper, we present an approach for the implementation of semantically enriched services in Enterprise Application Integration (EAI). We present an integration platform based on a Service Oriented Architecture (SOA) which consists of a service registry, a process designer and a run-time engine. There are some additional components for realizing semantic enrichment of services and composed processes 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 data transformations are assigned to each service during the deployment. The standard services of ERP, CRM, SCM etc. systems are encapsulated into mediator services which makes possible to apply them in a semantic integration framework. Still, created semantic services remain compatible with current Web service standards and communicate with standard SOAP messages. Hence the collaborative processes composed by attached semantic meta-information of services are also executable by standard Business Process Execution Language (BPEL) run-time engine.

Key-Words: - Enterprise application integration, 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 [1]. Spreading of such systems indicated the need for exchanging data and realizing communication between them. Thus companies strongly focus on business collaboration via enterprise application integration [2].

A lot of current approaches in enterprise application integration (EAI) are built 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. Some paper presents, that services are interfaces, some defines them as programming objects and some consider them as complex functions. In our case let say, that services represent exposed functionality of enterprise system and invoking a service means the execution of a function of an enterprise system.

Depending on the goal, the participating applications and the services in a specific case, different integration scenarios can be defined. There are service requesters and service providers in a specific SOA integration scenario. Requesters invoke services offered by the providers what means the specific execution of a function, e.g. a query, a calculation or the simple posting of data.

The solution of an integration scenario is one or more composed process. The processes consist of services, links connecting services and rules defining operations on process data or service execution order. For example in the simplest case the process is only a chain of services (services are invoked in a pre-defined order, after one another). The participating enterprise systems can be service requesters, service providers or even both in a composed process.

Besides building processes out of services can solve integration problems we can also create new capabilities on existing services of applications this way. A composed process can offer complex functionality which can be adopted in further composed processes as a single service.

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: on the analogy of the self-describing entities of the semantic web services can also provide meta-information about their type, compatibility, capability etc. 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 also help business consultants by designing collaborative business processes.

Hence semantic enrichment of process composition and services can have a two-fold focus: exploiting the advantages of semantic web technologies (well structured meta-data description of concepts and entities, strong reasoning capabilities, etc.) and help business consultants to recognize business entities originating from different systems in different format.

There are numerous current proposals which offer full approach e.g. methodologies and working tools for supporting the semantic extension of services. For example in [5] you can find a possible solution for attaching semantic meta-data to Web services by applying an improved WS-Policy (WS-Policy4MASC) standard.

In most of the current approaches 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 in this environment, our newly composed process possibly 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 the services remained intact. Thus there is a certain semantic gap between the description of input and output data schema of the service and its semantic description. For example we can have meta-data about the service defining that the service consumes data of address type. The address type is defined in our semantic repository (Ontology) as a data set consisting of country, town, ZIP-code, street and number data fields. However our standard service can provide this information in a different representation format, e.g. street and number can be contracted into one field separated by a special character. In this case applying our service in a composed process data type mismatch could be occurred by simply invoking it with an address data type. The same semantic distance can exist between services, where address output of a service can't be directly consumed by a service awaiting an input in another format of the address type. In spite of this, the attached semantic information referencing to the address concept is correct in both cases because this is the corresponding real world concept.

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. In our approach we show how to attach semantics to services, how to discover them and provide a lot of easy to use tools to realize it all. However this paper mainly focuses on the automatic creation of directly invokable proxy services for the process run-time.

The rest of the paper is organized as follows: Section 2 compares our approach with related work. Section 3 briefly describes our methodology for designing 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 contains evaluation of the results. Section 6 draws conclusions and outlines future work.

2 Related work

There are numerous researches presented in the literature about interoperable systems in service oriented architecture. For example, see [6, 7, 8, 9, 10, 11].

Solutions are adapted to many kind of field of application, e.g. digital multi-media [12] or production systems [13].

There are many tools and solutions to design and run standard BPEL processes, for example the Oralce Fusion Middleware [14] or the IBM Websphere [15]. 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. On the other hand, adding semantics may exclude the pure using of such standard run-time environments.

The Intelligent Software Agents Lab [16] at Carnegie Mellon deals with a SOA approach which is using Web Ontology Language for Services (OWL-S) semantic description for choreography.

They also developed tools for supporting the creation of the OWL-S description from a Web Service Description Language (WSDL) [17] basis and publicating them into a UDDI [18] registry.

In [19] the authors apply OWL-S to describe service behavior as well. The attached semantic description makes possible to discover services in the presented agent based framework. Services are found and ranked by the expected capability contained in the consumer's query and the matching algorithm involves ontological matching and evaluation of process constraints.

Instead of OWL-S we use BPEL to describe processes. In the last 2 years BPEL became the de-facto 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 Semantic Annotations for WSDL (SA-WSDL) for describing the services it was also possible to add semantic extensions and keep services in a BPEL compatible format at the same time.

In [20] the authors create a methodology and framework to compose processes by dynamic re-binding participating services. The processes are built of abstract services first. To find the proper binding to abstract services (in other worlds to find their equivalent invokable pair) a genetic algorithm is applied. Similar to our proposal proxy services are generated automatically in deployment time to encapsulate existing invokable services, but binding of concrete services is only done in run-time. In contrast to this we create fully invokable encapsulated proxy services already in deployment time what excludes the necessity of further central components (e.g. service registry) and methods (e.g implementing the binding part of services) in run-time. Furthermore providing transformation creation tools, and automatic adaptation of these rules into the encapsulated services, we also support the bridging of semantic gaps between different services of several vendors. In real world integration scenarios the participating services mostly provide their input and output information in different format. This implicates the necessity of applying such data mediation techniques during the service encapsulation.

In [21] the authors already apply transformations for data mediation. However this is done in process run-time. Hence it requires a very high reliability of the central units – a VieDAME 4 WS-BPEL environment containing a monitor, a selector, a transformer etc. unit in run-time as well. Our solution does not require additional installed units and special environment. The encapsulated services can be invoked and handled by standard soap messages of BPEL run-time engines the same way as standard web services. Furthermore, the new, mediated (proxy) service can also be hosted at the same place and the same way as the original (encapsulated) service is hosted.

Grossmann et al. [9] 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 and 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 [22]. Fig. 1. shows our simplified system architecture. The implemented tools are shown on the right and the composed process on the left side of the figure. Directly co-operating tools are also connected by lines.

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 of a customer as an input field, the address field of the customer provided by another service cannot be used.

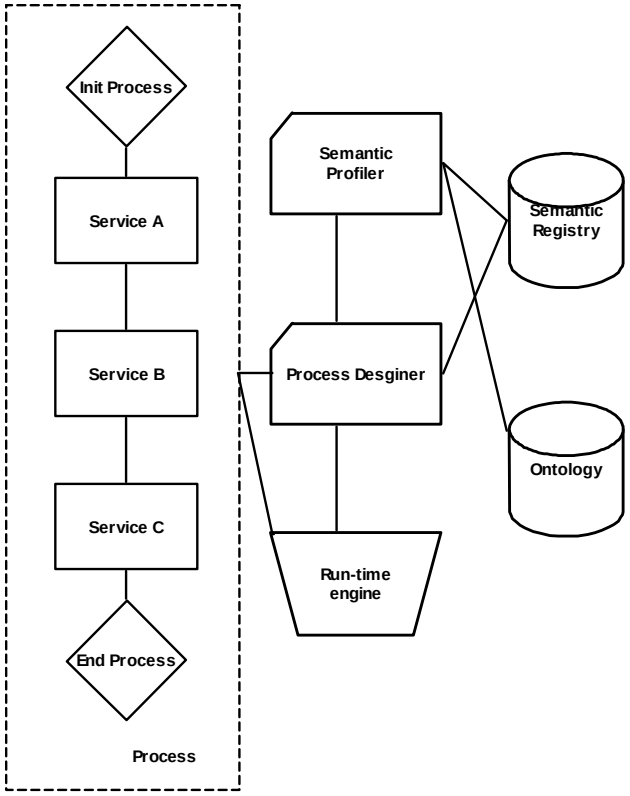

Fig. 1 Our integration architecture

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 [23].

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 [22].

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 component supports the identification and association of appropriate concepts e.g. service capabilities, input/output data types, service's pre- and poststates to services. 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 transform global schema data input of the semantic service to 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 problems e.g. the semantic gap between standard and semantic services, the transformation creation and the invokable proxy service 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 providing value for us 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. The proxy service:

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 is 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 invokable mediator services: modelReference extension attributes on certain elements of the WSDL document, which were designed and inserted by using the Semantic Profiler tool. These references connect the simple syntactic data types with the ontology level concepts, and points to 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 data-mediation.

Our service annotation methodology [23] defines the 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.

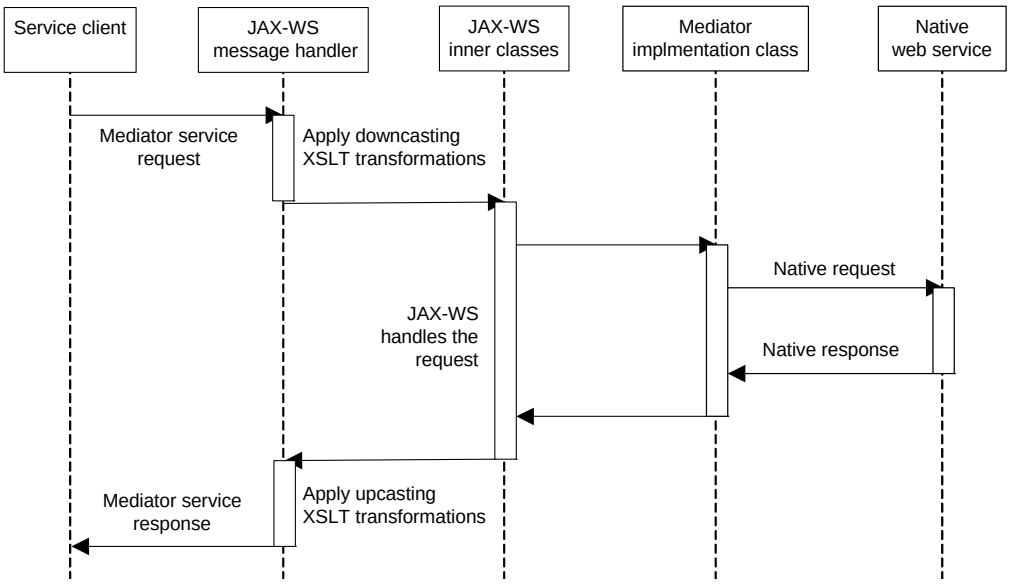

Fig. 2 Mediator service request-response sequence diagram

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, XSL Transformation (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 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 Web Ontology Language (OWL) is used. 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 an earlier identified 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 identified OWL 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 merged set of definitions is included in the input and output data type 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 [17] 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 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 (the algorithm was just presented above is used),

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).

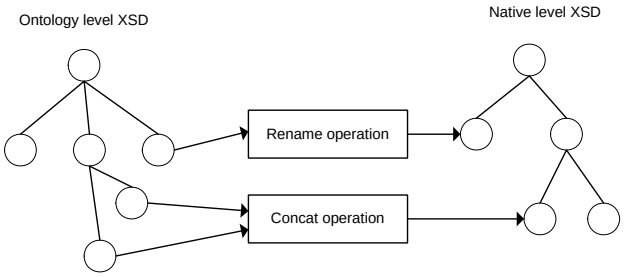

Fig. 3 Transformation example

4.2 Defining Transformations

The role of transformations was 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. Source and target schemas (trees) are shown on the left and the right side of the figure, and defined transformations are placed between them. The boxes represent transformation operations and are connected with the corresponding input and output nodes.

As you can see, the operations have only one output, and (may) have multiple inputs. For a simple case e.g. a string concatenation (shown at the bottom of the figure), let's imagine that the ontology level concept 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 the

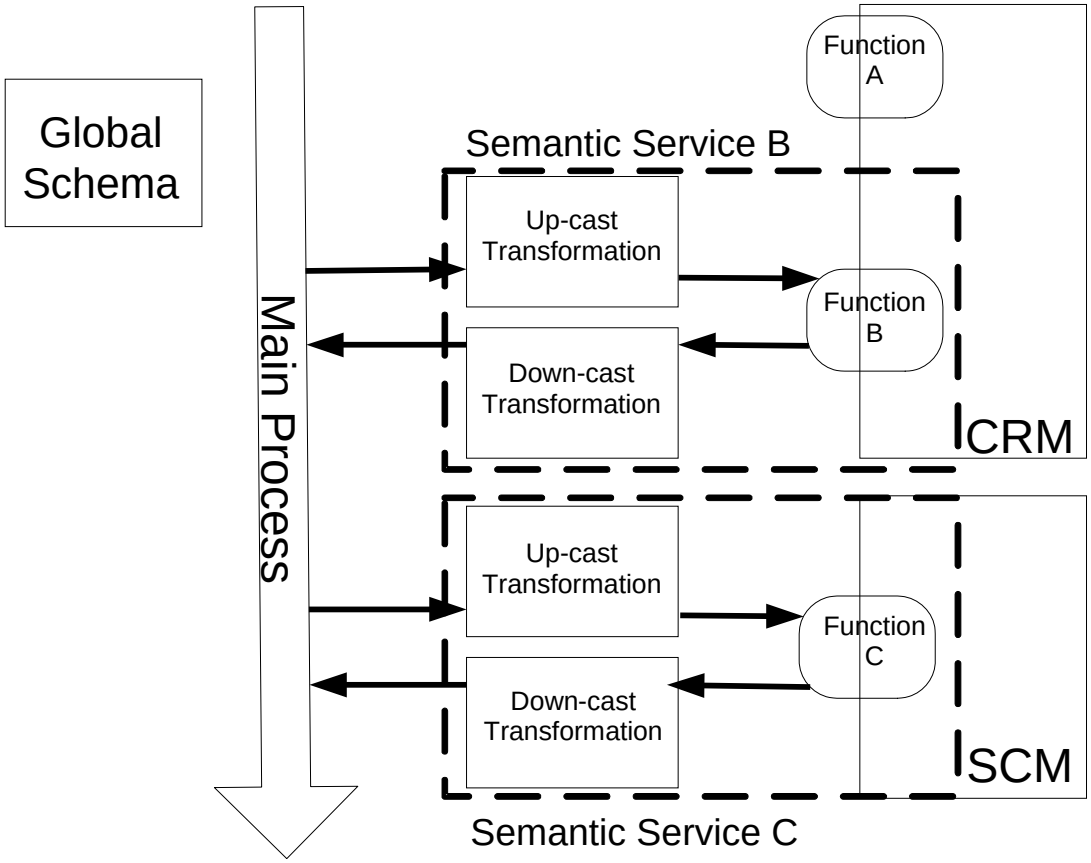

Fig. 4 Implemented process

properties of an ontology level class. It is possible, that on the native level only name, address and type, names are used respectively. So the value of the element must not be changed, but the tag names change, so that a rename transformation operation should be applied.

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.

5 Evalutation 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 seconds with the load of the ontology from a web URL, the service generation itself is 7-8 second and the build (running Ant) is another 4-5 seconds. But these have to be done only once for one version during the design time.

The availability and the efficiency of composed processes depend highly on their response time what are mainly influenced by the response time of the orchestrated mediated services. It is trivial, that we can not avoid the time costs coming from invoking standard services of ERP, CRM, etc. systems because they provides the business logic of existing information systems. However it is important to know about the extra time costs which we have to pay for applying semantically enriched (mediated) services also in run-time.

There are certain tasks e.g. performing transformations, catching and relaying messages, etc. by invoking a mediated service which takes some time; see in chapter 4 detailed. To analyze this time costs in run-time several experiments were implemented in our test environment. First the normal (not loaded) case was evaluated: we invoked the standard (native) and the mediated (proxy) service in every 3 seconds and compared the response times. The average was about 30-40 milliseconds for the native service and 200-300 milliseconds for the corresponding mediated service. There was a higher distribution in the measured values in the mediated case and an initiation phase is also observable. After the initiation phase the response time tends to 200 milliseconds. See detailed results in figure 5.

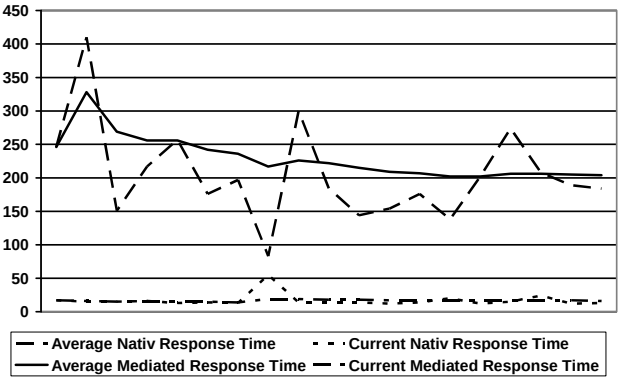

Fig. 5 Response times in normal (not loaded) case

For applying semantic services both in process composition and run-time we paid an extra 160-170 milliseconds by each service invoke. This difference is probably not disturbing for a human user of the system and comparing it with the possible savable time at the process composition it is a fair overhead in the run-time.

After evaluating this simple case more realistic, loaded cases should also be evaluated. By designing hardware requirements and sizing systems it is important to know how the framework responds on greater loads of concurrent request. To simulate this 10, 20 and 30 concurrent users were posting service request to our system. The requests were also timed after two different strategies:

• by *constant load* every client invoked the service after every 500 millisecond and

• by *burst load* every client invoked the service continuously (after the services returned with the

response it was requested again immediately) for every first 20 seconds in every minute.

To evaluate the results of our run-time architecture the experiments were done both for native and mediated services and the response times of these were compared again. All tests were run for 120 seconds long. Figure. 6 (7) shows the results for 10 (20) concurrent clients by constant load.

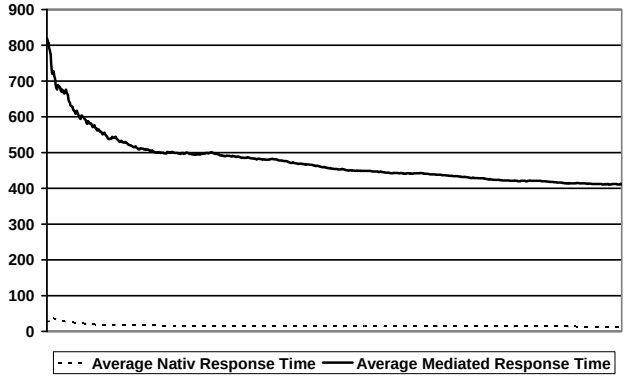

Fig. 6 Response times for 10 clients by constant load

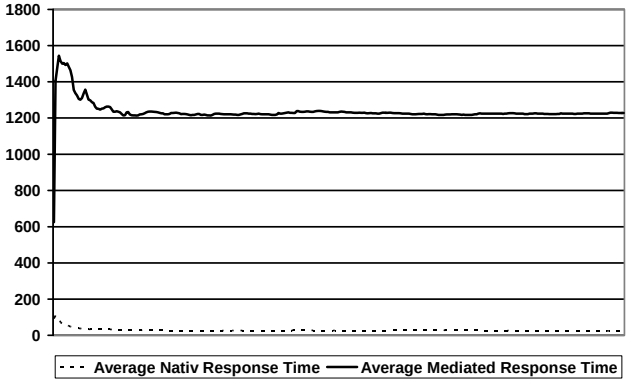

Fig. 7 Response times for 20 clients by constant load

Figure 8 (9) shows the results for 10 (20) concurrent clients by burst load.

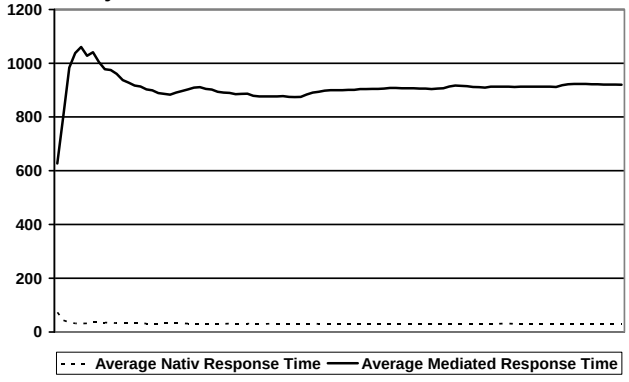

Fig. 8 Response times for 10 clients by burst load

The results of the 30 client cases follow the trends of the 10 and 20 client cases. Thus we only present them by the measured values without attaching the diagrams.

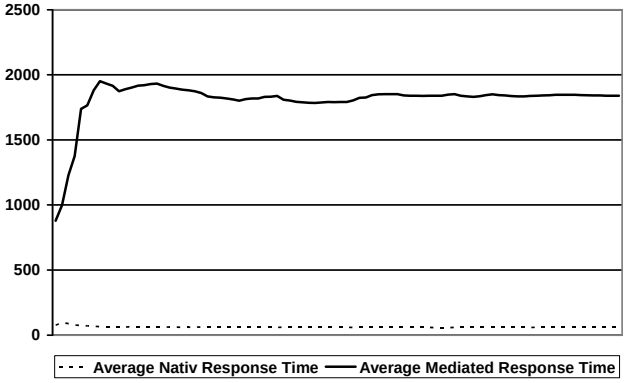

Fig. 9 Response times for 20 clients by burst load

At the constant load cases the average response time stabilized around 15 (24, 30) milliseconds for 10 (20, 30) clients by invoking the natives service. These values were 500 (1200, 2300) milliseconds for the mediated service. This shows that the mediated architecture is not as scalable as the original. This incident can be explained by the applied hardware infrastructure, which has reached its physical limits only in the mediated case. This is also proved by the results of the burst cases, where the measured values are promising. At the burst load cases the average response time stabilized around 30 (60, 90) milliseconds for 10 (20, 30) clients by invoking the native service. These values were 900 (1800, 2900) milliseconds for the mediated service. Because the response times increase proportionately the system remains scalable also for the mediated services. Furthermore the response times for such a high load were not so high in spite of that the tests were only run on a standard home computer.

6 Conclusions

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. Furthermore advanced verification methods and testing strategies could be also applied in our framework like the one presented in [25]. Finally, 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.

The performance of the process run-time environment could also be evaluated and increased with applying custom run-time solutions for example similar to the approach presented in [26]. 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:

[1] A.-W Sherr, *Business Process Engineering - Reference Models for Industrial Enterprises*, Springer-Verlag, Berlin, 1994.

[2] Q. Ni, W. F. Lu, K.D.V. Yarlagadda, X. Ming, A collaborative engine for enterprise application integration, *Computers in Industry*, vol. 57, 2006, pp. 640–652.

[3] D. K. Barry, Web services and service-oriented architectures, *Service-Oriented Architectures and Web Services*, 2003, pp. 17-33.

[4] P. Martinek, B Szikora, Semantic Execution of BPEL processes, *Proceedings of the 16th International conference on Information System Development (ISD2006)*, Budapest, Hungary, 2006, pp. 361-367.

[5] Tosic, V., Erradi, A., Maheshwari, P., On extending WS-Policy with specification of XML Web service semantics, monitoring, and control driven by business value, *WSEAS Transactions on Computers*, vol 6, issue 5, 2007, pp. 805-812

[6] S. Arroyo, M-A. Sicilia, J-M. Dodero, Choreography frameworks for business integration: Addressing heterogeneous semantics, *Computers in Industry*, Volume 58, Issue 6, 2007, pp. 487-503.

[7] M-A. Barbosa, L-S. Barbosa, Configurations of Web Services, *Electronic Notes in Theoretical Computer Science*, Volume 175, Issue 2, 2007, pp. 39-57.

[8] M. Chen, D. Zhang, L. Zhou, Empowering collaborative commerce with Web services enabled business process management systems, *Decision Support Systems*, vol. 43, 2007, pp. 530– 546.

[9] G. Grossmann, Y. Ren, M. Schrefl, M. Stumptner, Behavior Based Integration of Composite Business Processes, *Business Process Management*, Springer Berlin/ Heidelberg, 2005, pp. 186–204.

[10] I. Navas-Delgado, M.M. Roldán-García, J. F. Aldana-Montes, Kreios: Towards Semantic Interoperable Systems, *Advances in Information Systems*, vol. 3261, Springer Berlin/Heidelberg, 2004, pp. 161–171.

[11] Georgiou, L., Pyrovolakis, O.I, Designing web services for business processes through integrated development environments, *WSEAS Transactions on Computers*, vol. 4, issue 8, 2005, pp. 908-916

[12] Yun, S.-B., Ko, H.-J., Kim, U.-M., The design and the implementation of web service security system for the secured distribution of digital contents, *WSEAS Transactions on Computers*, vol. 5, issue 2, 2006, pp. 348-351.

[13] Marius, C., Lucian-Ionel, C., Sabin-Corneliu, B., Using semantic web technologies to improve the design process in the context of virtual production systems, *WSEAS Transactions on Computers*, vol. 4, issue 12, 2005, pp. 1788-1793.

[14] Oracle Fusion Middleware, at http://www.oracle.com/technology/ products/middleware/index.html viewed 10.07.2006.

[15] IBM Websphere at http://www-306.ibm.com/software/websphere/ viewed 10.07.2006.

[16] Intelligent Software Agents Lab, at http://www.cs.cmu.edu/~softagents/ viewed 10.07.2006.
[17] W3C, Web Service Description Language, *W3C Working Group Note*, at http://www.w3.org/TR/wsdl, 2001.03.15.
[18] OASIS, UDDI, Advanced Web Services Discovery Standard at http://www.uddi.org/specification.html, 2004
[19] Sriharee, N., Senivongse, T., On matching and ranking of web services behaviour in process-based service discovery, *WSEAS Transactions on Computers*, vol. 5, issue 2, 2006, pp. 439-446.
[20] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, Maria Luisa Villani, A framework for QoS-aware binding and re-binding of composite web services, *Journal of Systems and Software*, vol 81, issue 10, 2008, pp. 1754-1769
[21] Oliver Moser, Florian Rosenberg and Schahram Dustdar, Non-Intrusive Monitoring and Service Adaptation for WS-BPEL, *Proceedings of the 16th International World Wide Web Conference (WWW 2008)*, Beijing, China, 2008, pp. 815-824.
[22] P. Martinek, J. Kerekes, B. Szikora, Semantically-enriched Service-Oriented Business Applications, *Proceedings of the 29th International Spring Seminar on Electronics Technology (ISSE2006)*, Dresden, Germany, 2006, pp.
[23] P. Martinek, B. Tóthfalussy, B. Szikora, Semantically Described Services in the Enterprise Application Integration, *Proceedings of the 30th International Spring Seminar on Electronics Technology (ISSE 2007)*, Cluj-Napoca, Romania, 2007, pp. 335-338.
[24] Visigoth Software Society - The Freemarker template engine at http://www.freemarker.org/, viewed 2008.03.01.
[25] Lertphumpanya, T., Senivongse, T., Basis path test suite and testing process for WS-BPEL, *WSEAS Transactions on Computers*, vol. 7, issue 5, 2008, pp. 483-496.
[26] Cesare Pautasso, Thomas Heinis, Gustavo Alonso, Autonomic resource provisioning for software business processes, *Information and Software Technology*, vol. 49, 2007, pp. 65-80.