DS^{3}I - A Dynamic Semantically Enhanced Service Selection Infrastructure

Christoph Fritsch[1], Peter Bednar[2], and Günther Pernul[1,⋆]

[1] Department of Information Systems
University of Regensburg
93053 Regensburg, Germany
{christoph.fritsch,guenther.pernul}@wiwi.uni-regensburg.de
http://www-ifs.uni-regensburg.de

[2] Centre for Information Technologies
Technical University of Kosice
04200 Košice, Slovakia
peter.bednar@tuke.sk
http://www.ekf.tuke.sk

Abstract. Service-oriented computing (SOC), lately often in combination with business process modeling (BPM), is becoming reality in current enterprise systems. At the same time, process models do no longer only span different departments but more and more frequently multiple organizations. Current BPM approaches focus on syntactic specification of work flow structures and necessitate static service allocation during modeling time. Dynamic service allocation at runtime based on the semantics of service descriptions, however, allows for much more flexibility. This paper presents DS^{3}I as a concept and implementation for semantically enhanced dynamic service selection. DS^{3}I and its semantic middleware components are based upon an ESB and semantic annotations of services and allows for one-step service selection at runtime of a process.

Keywords: Dynamic Service Selection, Semantic Mediation, Global Service Infrastructures.

1 Introduction and Motivation

Today's companies are facing strong challenges in the markets. More and more companies are realizing that they can no longer operate as fully self-contained actors but have to advance their ways of doing business. They begin to reconsider their business structures and aim for flexible cross company business network structures to perform future business with anybody, anywhere, anytime[18]. Two major trends accompany this development: (1) *Process Modeling*, i.e. the investigation and structured graphical presentation of business processes (2) *Service Composition*, i.e. chaining and automated execution of services. Combined both

⋆ The research leading to these results received funding from the European Community's Seventh Framework Programme under grant agreement no. 217098.

technologies allow for automatic execution and control of business processes. The common approach to obtain such a executable process model is to first break down the overall process into single tasks, then assign appropriate electronic self-contained services and finally map their inputs and outputs. The allocation of services to tasks is static. The resulting executable process model together with references to all external services is then deployed to a work flow engine (WFE).

Consider for example an online shop that receives orders via a web application and passes order processing over to a WFE that executes BPEL processes. Order processing presumes communication with partner services for example for credit card verification or payment processing. Even if this kind of service is available from various providers, outage time or failures of the external services cause interruption of the overall order processing due to static assignment of partner links in the BPEL process. Thus service allocation at process modeling time brings along undesirable limitations: (1) It is susceptible to failures due to unavailable external services. Overall process execution fails if a single partner link is not available. (2) Newly published services cannot be considered in the process model without altering and redeploying it. Process modeling as the basis of controlled execution is essential but the trend towards intra- and inter-organizational service orientation necessitates more flexibility. Appropriate services have to be allocated at the latest possible point in time, i.e. at runtime of the process.

In this paper we present DS^{3}I as an approach for dynamic service selection and mediation as it is developed as part of the SPIKE project[1]. Using DS^{3}I process models do no longer (necessarily) establish static links between tasks and service instances but are rather built against generic service interfaces. Behind these facades, semantically enhanced dynamic service selection and allocation is performed. We particularly focus on non-intrusive dynamic service allocation, on the semantic description and resolving of services and the semantic and non-semantic mediation. DS^{3}I strives for suitability for legacy applications as 'real services' are just evolving in many organizations whereas thousands of legacy applications prove aptitude in everyday life.

The remainder of the paper is organized as follows: First we provide provides related work in Section 2. Section 3 summarizes the conceptual model before Sections 4 and 5 present the proposal in detail on the architectural level and implementation details of a prototype system, respectively. Finally, Section 6 draws conclusions and identifies current and future work.

2 Related Work

Schmidt et al. [16] noticed already back in 2005 that "SOA holds out the promise that services can be discovered [...] and bound together to form new and exciting, or simply more efficient applications". They developed two generic patterns: (1) The protocol switch pattern and (2) the service transformation pattern. Both

1 `http://www.spike-project.eu`

allow for discovering suitable target services dynamically. However, as these patterns have not yet been broadly implemented, service discovery is still carried out before a client is developed or the business process is modeled and deployed. Most related and decisive approaches for dynamic service selection are presented below and improvements by semantic annotations are presented in section 2.3.

2.1 Degrees of Freedom

Both Alonso et al. [1] and Chang et al. [4] provide comprehensive classification schemes which uncover the main challenges. In combination both approaches cover the essential degrees of freedom for dynamic service selection approaches. Alonso et al. [1] distinguish between four concepts: (1) *Static Binding* is the concept current process modeling approaches , i.e. static allocation of services at modeling time. (2) *Dynamic Binding by Reference* depends on a variable defined in the process which contains a reference to the chosen service instance. (3) *Dynamic Binding by Lookup* employs service repositories to retrieve predefined services at runtime. Finally, (4) *Dynamic Operation Selection* in the authors grasp is a rather special case that deals with selecting different operations of the same service.

This classification focuses on service selection as part of the overall process and does not consider unequal responsibilities for modeling and service allocation and/or the need to modify service allocation criteria without modifying the process. Nevertheless, they realize the necessity to separate between process modeling and service allocation time. Chang et al. [4] provide a different classification of dynamic composition. They classify along the axes “decision time", i.e. the moment the decision is made , “target service visibility", i.e. if the set of service candidates is fixed or time-varying, and “provision of adaption method", i.e. if the target service can adopt to given service requests.

2.2 Dynamic Service Selection

Content-based routing (CBR) in Enterprise Service Buses (ESB) as mentioned by [15] and others is the most mature approach. Based on a service request messages' contents a component decides to which service instance a request is forwarded. DRESR [2] by Bai et al. introduces the idea of abstract routing paths (ARPs), where each service is identified by an URI and an abstract service name. An ARP is composed of abstract service names and is therefore not bound to particular service instances. To obtain concrete service instances, a central routing manager component selects a service instance from the pool of candidates for the given task. The Dynamic Wire Tap (DWT) approach by Wu et al. [20] builds upon the well-known EAI patterns Wire Tap, Enricher, Recipient List and Aggregator [8]. A service discovery engine retrieves a list of service candidates and forwards the request to the DWT component which in turn routes the request forward to one of the service candidates. CBR and DRESR, however, do not offer real dynamic service selection as the potential target services together with routing rules have to be statically defined beforehand and DWT depicts a rather cloudy low-level concept and demands for quite some adoptions.

Other approaches consider BPEL as a starting point: WS Binder [5] by Di Penta et al. binds tasks to proxy objects instead of service instances. During a pre-execution binding the proxy objects are initialized with service instances resulting in a quasi-static service allocation. VRESCo by [14] and [12] introduces an aspect-oriented extension for BPEL environments for monitoring and replacing partner links. While WS Binder does not consider transformation/mediation, VRESCo at least considers static, non-semantic transformation rules. It does, however, neither distinguish between abstract and concrete service interfaces nor does it harness semantic meta-data therefore potentially resulting in a big amount of complex static transformation rules.

The concept of Dynamic Composition Handlers (DCHs) by Chang et al.[3] builds upon an ESB and clearly separates between service interfaces specified in the process and realized interfaces. It does, however, only consider a BPEL-engine as service client. The ESB-based ProBus approach by Mietzner et al. [13] is a concept for policy-driven dynamic service selection. In contrast to other work, this approach mainly focuses on the definition of selection criteria. Following the ProBus idea, service requesters define their preferences in form of non-functional properties expressed as WS-Policy statements and the services are described by WSResourceProperties. ProBus matches the service requester's policy against resource properties of known services to obtain a service candidate.

2.3 Semantic Technologies

Improvements in the field of dynamic service selection gained from semantic technologies can be divided into two blocks: (1) Semantically annotated services and process models and (2) semantically supported mediation and resolving.

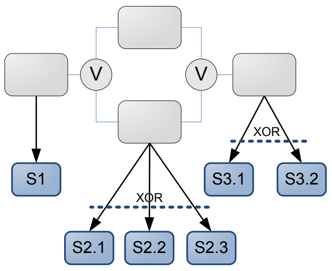
Semantically enhanced business process modeling and semantic ESB is in focus of several research initiatives, e.g. the Object Management Group or EU-fundend R&D projects such as STASIS or the SUPER project. Several rather mature but only scarcely used concepts exist. The *Resource Description Framework (RDF)* has primarily been designed as a meta-data data model for all kinds of (web) resources has become a standard semantic model for data interchange. *OWL-S* offers an ontology of services and aims at making semantic description, discovery and execution of services possible. The *Web Service Modeling Ontology (WSMO)* emerged as a result of several EU-funded research projects. Its main concepts are ontologies, web services, (service) goals and mediators. Finally, Semantic Annotations for WSDL and XML Schema *(SAWSDL)* forms a simple extension layer on top of WSDL that lets components specify their semantics.

Semantic process and service annotation alone does only provide the foundations for semantically enhanced dynamic service selection. As can be seen from the previous section, a number of authors pay attention to non-semantic dynamic service selection but only sporadically complete and comprehensive semantic approaches are published. Karastoyanova et al. [9] proposed a reference architecture for semantic business process management where they clearly distinguish between a process modeling and runtime environment. The reference architecture has later been implemented as an semantic service bus [10]. Fujii and

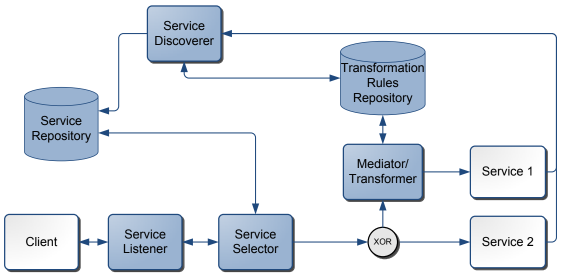
Suda [7] present another comprehensive framework for semantics-based dynamic service composition. The framework does neither embrace an prototype nor does it build upon an modeled processes and an ESB as infrastructure component. It considers user context information for service composition and allows users to formulate a request for an particular type of application in natural language.

3 Conceptual Model

The conceptual model of DS^{3}I is sketched in Fig. 1(b). The essential goal is dynamic service selection in a minimum-invasive way. The switch from static to dynamic service allocation shall not necessitate any modifications. Neither service requester, be it a stand-alone client application or a WFE, nor individual service instances shall be required to be modified. Instead, infrastructure components allow for dynamic service allocations.



(a) Static Service Allocation

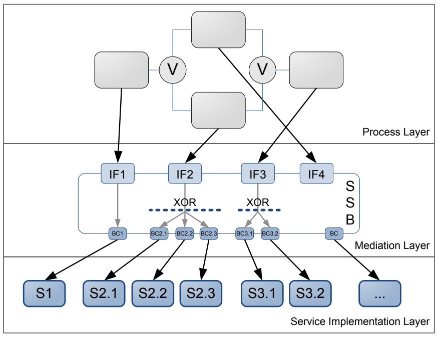


(b) Service Selection and Mediation Scheme

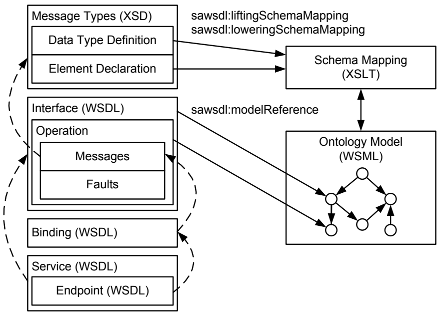
Fig. 1. Static vs. Dynamic Service Allocation

Consequently, the conceptual model in Fig. 1(b) consists of two different types of components: (1) The white shaded actors depict unmodified parties that request or provide a given service. (2) The bluish shaded components depict components that facilitate the semantically supported dynamic service selection approach. These components are shielded by the infrastructure and are therefore neither visible for client nor service provider.

Details on components' functionality and implementation are illustrated in sections 4 and 5 so only a short overview is presented here: The *Service Listener* acts as a virtualized interface for a given type of service. We assume that service request messages are dispatched to these virtual services instead of concrete ones. Based on the request message and further selection criteria and non-functional properties, the *Service Selector* picks an appropriate service instance from the Service Repository. The *Service Repository* is charged with information and descriptions of available services via the *Service Discoverer* which in turn provides means for service providers to announce their services. As different services that provide the same functionality may vary in their interfaces and message format, the *Mediator/Transformer* mediates between clients' request messages and the



(a) 3-layered Operating Mode



(b) Semantic Annotation of Services

Fig. 2. Operation Mode and Semantic Annotations

request messages expected by the selected service instance. It retrieves applicable transformation rules from the *Transformation Rules Repository* and transforms messages accordingly. We intentionally designed a comprehensive *infrastructure*. This way service selection can be delegated completely to the infrastructure, ensuring clear separation between different roles such as process modelers, service providers or service users. Applying the conceptual service selection model to current service binding concepts results in a 3-layered operating mode illustrated in Fig. 2(a). The three layers are organized as follows:

- The *Process Layer* focuses on modeling the business process and its stepwise refinement. Furthermore, process deployment to a WFE as well as execution monitoring is conducted at this layer. Different tasks of executable processes are linked to virtual service interfaces provided by DS^{3}I.
- The *Mediation Layer* forms the core of DS^{3}I. It provides generic interfaces (IF1 through IF4 in Fig. 2(a)) against which the process is linked and holds links (BC1 through BC4 in Fig. 2(a)) to all available service instances that form the pool of service candidates. As a result, the mediation layer mediates between virtualized service interfaces and concrete service instances.
- Actors on the *Service Implementation Layer* implement inquired functionality in form of services. Each service instance realizes an interface which is described and registered with DS^{3}I.
- A *Management Layer* is orthogonal to the other layers and therefore not displayed in Fig. 2(a). At this layer the service instances are announced to the mediation layer and the criteria and non-functional properties that determine the service selection procedure are defined.

4 Architecture

The overall architecture resulting from this conceptual model is depicted in Fig. 3. It consists of a service requester, several interchangeable service candidates and an extended enterprise service bus as a semantically supported dynamic service selection infrastructure.

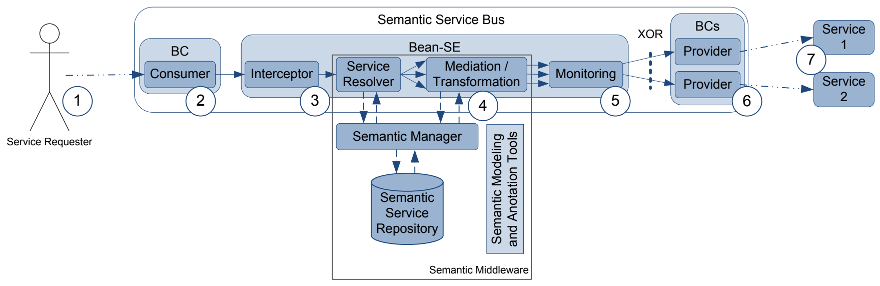


Fig. 3. DS^{3}I Architecture Overview

4.1 ESB - A Means to an End

DS^{3}I builds upon standards ESB capabilities. An ESB acts as an intermediary between service requesters and service instances and shall provide diverse messaging, routing and mediation capabilities [11,15] as defined in the Java Business Integration (JBI) standard [17]. A JBI compliant ESB consists of three components DS^{3}I is built upon:

– *Binding components (BCs)* allow connecting external services via various communication protocols and transforming data to a normalized form. Using existing BCs we employ the JBI-standard to interlink DS^{3}I not only with web services but with diverse kinds of legacy applications which were only seldom designed to be linked together.
– *Normalized Message Router (NMR)* forms the central messaging and routing backbone of the ESB, and therefore the backbone where service selection components are hooked in.
– *Service Engines (SEs)* provide business logic for integration of services (i.e. orchestration, data transformation, etc.). All semantic-enabled components are implemented as JBI SEs.

4.2 Components and Capabilities

DS^{3}I can be broken down into basically five different technical building blocks. The semantic middleware, in turn, is composed of several more modules as depicted in the black-bordered box in Fig. 3.

– The *Service Requester (1)* can be any stand-alone client or WFE that invokes services. For the time being we assume service requesters to communicate via SOAP with DS^{3}I. There is no need for the service requester to be changed in terms of additional functionality before it can benefit from the dynamic service selection infrastructure.
– *DS^{3}I* acts as the communication and messaging infrastructure. It provides both message sinks and sources for service requesters and service providers which is why JBI BCs reside at both ends: On the client side one for each exposed virtualized service interface (2), on the service candidate side one for

each connected service candidate (6). Furthermore, it serves as the runtime engine for the semantic middle components.
- The *Semantic Middleware (4)* consists of run-time components for ontologies storage, maintenance, semantic mediation, validation and querying/reasoning over semantically described data on services and messages. In combination, these components provide all capabilities depicted in Fig. 1(b). It consists of service resolving components, which actually select an appropriate service instances for a given virtualized interface based on semantic annotations of services and their interfaces. In a subsequent step, mediation components transform request and response messages into an adequate format for the selected service instance, again based on semantic interface descriptions. Both components employ the semantic manager, which shields all semantic mediation and reasoning related capabilities. The semantic manger in turn resorts to the semantic repository where all required information is stored.
- Functionality and interfaces of *Service Candidates* need to be semantically annotated. This is supported by *Semantic Modeling and Annotation Tools* for knowledge engineers and annotators at service provider side. Design tools are available as a set of Eclipse plug-ins and allow for visual modeling of ontology elements together with semantic annotations of WSDL files.
- Individual *service candidates* are realized by any kind of service implementation which is supported by a JBI BC. Hence, DS^{3}I is not restricted to SOAP-based webservices but can interlink with a broader set of service implementations. Merely, the provided functionality and their inputs and outputs have to be annotated semantically via previously mentioned tools.

The architecture involves two more auxiliary components: The *Message Interceptor (3)* allows for easily enabling or disabling the semantic middleware. Here a client-sided BC can easily be configured and statically be bound to a fixed service instance to circumvent the semantic middleware in case of need. The *Monitoring Component (5)* aims at collecting non-functional properties for each interaction with a service instance. Resulting data may provide a basis for both service evaluation and monitoring as well as for future service selection decisions.

5 Implementation

To evaluate the conceptual model and the DS^{3}I architecture we implemented and tested the individual components prototypically within the SPIKE project.

5.1 Implementation Considerations

As semantic framework we chose WSMO-Lite for handling ontologies and semantically enriched data. In particular, the implementation is based on the following components and frameworks:

- The *SPIKE API* for in-memory representation of the ontology elements (ontologies, concepts, instances, relations and axioms) is based on the wsmo4j library. Besides the ontology API, wsmo4j provides facilities for ontology validation and parsing/serialization from and to various formats.

- The main functionality of the *Framework for RDF persistence* is the mapping of top ontology elements into the RDF model. SPIKE RDF persistence is based on the ORDI framework, which allows integrating various data sources and provides a common RDF API for accessing underlying data.
- *RDF storage.* SPIKE ontologies are physically stored as RDF data using the Sesame repository. The current SPIKE configuration uses SwiftOWLIM extended by the TRREE inference engine as physical storage.
- For *infrastructure components* (external service interfaces, runtime environment for service selection and mediation components) the JBI-compliant ESB Apache ServiceMix and its various BCs and SEs are employed. For the implementation of virtualized service interfaces as well as for the links to external services the Apache CXF BC is used. As runtime environment for the sementic middleware the ServiceMix Bean SE is used which allows deploying Java classes into the ESB. Semantic (and non-semantic) dynamic service selection components are thus implemented as plain Java beans.
- *Semantic Annotation of Services* is implemented using the SAWSDL specification as shown in Fig. 2(b). SAWSDL extensions take two forms:
  1. Model references (`sawsdl:modelReference` attribute) which point to semantic concepts by URIs. Model references can be applied to WSDL elements (i.e. interface or operation) to specify the function of the service or XML scheme elements and describe the meaning of the input/output data.
  2. Schema mappings (`sawsdl:liftingSchemaMapping` and `sawsdl:loweringSchemaMapping` attributes) specify data transformations (usually defined with XSLT) between messages in normalized XML format (as used by the ESB) and the associated semantic model. The schema mappings are used for semantic data mediation. An automated semantic mediator can first lift data in one XML format to instances in the shared ontology and then again lower it to another XML format using the lifting annotation from the first format's schema and the lowering annotation from the second schema.

5.2 Semantic Resolving and Mediation

Preconditions. Semantic annotations of services are used to overcome the ambiguities during service discovery related to the description of services at the syntactic level only. For service composition, we adopt a semi-automatic approach where the business processes are modeled manually as BPEL processes. BPEL processes refer to abstract partner links, the virtualized service interfaces. The abstract partner links have to be resolved to concrete service instances during process execution. The process of resolving can be automatic and is based on semantic matching of descriptions of abstract partner links and service candidates provided by potentially several service providers.

In order to overcome data heterogeneity (i.e. when data expected by the abstract partner link has a different format than data defined for the selected service instance), DS^{3}I supports semantic data mediation. Matching of service candidates is based on two types of semantic annotations assigned to the abstract partner link using the `modelReference` SA-WSDL attribute: (1) SKOS ontology [19] for specification of classification schemes of categories. (2) Domain specific semantic types of input/output messages specified for the requested operation.

During matching both types can be combined arbitrarily and the hierarchical organization of categories and subclass/superclass relations of input/output types can be recursively expanded during reasoning. For semantic mediation, we adopt two approaches. The first approach is based on standard XSLT where ontologies are used as the common data vocabulary. This approach requires XSLT transformations for lifting and lowering of instances and is well supported by existing SEs. In the second approach incoming XML data is transformed to instances using a generic lifting scheme. Input instances are then transformed using semantic mappings into output instances which in turn are transformed to XML data again using the generic lowering scheme.

Operation Sequence. The whole procedure for business process execution in DS^{3}I consists of the following steps:

1. A business process definition is deployed to the BPEL SE. BPEL process invokes of abstract partner links are sent to the service resolving component.
2. The Service Resolver calls the Semantic Manager for discovery of services capable to provide outputs as specified for the abstract partner link. If there are more service candidates, a target service is selected for invocation according to predefined properties.
3. In case of unequal data formats specified for the abstract partner link and the target service, the message is forwarded to the Mediator, otherwise it is forwarded directly through the JBI BC to the selected service instance.

During mediation, the message is processed as follows: (1) Input data from the message exchange is transformed using the lifting schema specified for the abstract partner link. The result is a set of semantic instances. (2) Since the domain ontology specified for the partner link can be different to the ontology specified for the resolved service, instances are optionally transformed from the source ontology to the target ontology using semantic axioms and transformation rules (instance-to-instance transformation). (3) Instances are transformed back to normalized messages using the lowering schema specified for the resolved service.

In summary, data is first transformed using the lifting schema of the resolved service and then transformed back to normalized messages using the lowering schema specified for the partner link.

6 Conclusions and Future Work

In this paper we have introduced DS^{3}I for semantically enhanced dynamic service selection and mediation. DS^{3}I allows for one-step service selection without any negotiation phase between clients and service providers. DS^{3}I and its semantic middleware components are based upon an ESB and employ semantic annotations of services. Semantic descriptions are furthermore applied to find appropriate service candidates and mediate between unequal interfaces and data formats. Except for more detailed service descriptions, DS^{3}I does not necessitate any changes at client or service side.

Employing DS^{3}I, appropriate service candidates can be discovered and assigned at run-time This way process modelers and developers are no longer required to statically allocate service instances already at modeling time of a business process or implementation time of a stand-alone client application and service instances published at a later date can still be considered. Process modeling gains much more flexibility and a clear separation between process modelers or client application developers and business operation personnel can be accomplished. The former ones can focus on functional and business-process requirements while issues of service selection and mediation are delegated to business operation personnel and infrastructure components. We presented a detailed problem breakdown together with related work in this area. The core area of this work, however, is the DS^{3}I conceptual model, the overall architecture of the semantically enhanced dynamic service selection infrastructure and details on the prototypical implementation.

While our work yielded a suitable prototype for semantically supported dynamic service selection and mediation, future work is divided into two areas of research. (1) Requirements definition for the service selection phase along with performance penalties due to the gained flexibility have to be investigated in detail. (2) Dynamic service selection presupposes dynamic security enforcement and despite dynamic allocation of services, access control and accountability have to be ensured. We already developed a proposal [6] which is being elaborated and tested. Finally, as service selection criteria may vary depending on the sender of the initial message, both previously mentioned fields of research need to be reintegrated to allow for extracting service selection criteria from predefined configurations and user profiles.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web S: Concepts, Architectures and Applications. Springer, Berlin (2004)
2. Bai, X., Xie, J., Chen, B., Xiao, S.: DRESR: Dynamic Routing in Enterprise Service Bus. In: Proc. of the IEEE International Conference on e-Business Engineering (ICEBE 2007), pp. 528–531 (2007)
3. Chang, S.H., Bae, J.S., Jeon, W.Y., La Jung, H., Kim, S.D.: A Practical Framework for Dynamic Composition on Enterprise Service Bus. In: IEEE International Conference on Services Computing, pp. 713–714 (2007)
4. Chang, S.H., La, H.J., Bae, J.S., Jeon, W.Y., Kim, S.D.: Design of a Dynamic Composition Handler for ESB-based Services. In: Proc. of the IEEE International Conference on e-Business Engineering (ICEBE 2007), pp. 287–294 (2007)
5. Penta, M.D., Esposito, R., Villani, M.L., Codato, R., Colombo, M., Nitto, E.D.: WS Binder: A Framework to Enable Dynamic Binding of Composite Web Services. In: Proc. of the 2006 International Workshop on Service-oriented Software Engineering (SOSE 2006), pp. 74–80 (2006)

6. Fritsch, C., Pernul, G.: Security for Dynamic Service-Oriented eCollaboration - Architectural Alternatives and Proposed Solution. In: Proc. of the 7th International Conference on Trust, Privacy & Security in Digital Business (TrustBus 2010), pp. 214–226 (2010)
7. Fujii, K., Suda, T.: Semantics-based context-aware dynamic Service Composition. ACM Transactions on Autonomous and Adaptive Systems (TAAS) 4(2), 1–31 (2009)
8. Hohpe, G., Woolf, B., Brown, K.: Enterprise integration patterns: Designing, building, and deploying messaging solutions. Addison-Wesley, Boston (2008)
9. Karastoyanova, D., van Lessenand Frank Leymann, T., Ma, Z., Nitzsche, J., Wetzstein, B., Bhiri, S., Hauswirth, M., Zaremba, M.: A Reference Architecture for Semantic Business Process Management Systems. In: Multikonferenz Wirtschaftsinformatik (2008)
10. Karastoyanova, D., Wetzstein, B., van Lessen, T., Wutke, D., Nitzsche, J., Leymann, F.: Semantic Service Bus: Architecture and Implementation of a Next Generation Middleware. In: Proc. of the 2nd International ICDE Workshop on Service Engineering (SEIW 2007), pp. 347–354 (2007)
11. Leymann, F.: The (Service) Bus: Services Penetrate Everyday Life. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 12–20. Springer, Heidelberg (2005)
12. Michlmayr, A., Rosenberg, F., Leitner, P., Dustdar, S.: End-to-End Support for QoS-Aware Service Selection, Binding, and Mediation in VRESCo. IEEE Transactions on Services Computing 3/2010, 193–205 (2010)
13. Mietzner, R., van Lessen, T., Wiese, A., Wieland, M., Karastoyanova, D., Leymann, F.: Virtualizing Services and Resources with ProBus: The WS-Policy-Aware Service and Resource Bus. In: Proc. of the 7th International Conference on Web Services, ICWS 2009, pp. 617–624 (2009)
14. Moser, O., Rosenberg, F., Dustdar, S.: Non-intrusive monitoring and service adaptation for ws-bpel. In: Proc. of the 17th Int. Conference on World Wide Web, WWW 2008, pp. 815–824. ACM, New York (2008)
15. Papazoglou, M.P., van den Heuvel, W.-J.: Service oriented Architectures: Approaches, Technologies and Research Issues. The VLDB Journal 16(3), 389–415 (2007)
16. Schmidt, M.-T., Hutchison, B., Lambros, P., Phippen, R.: The Enterprise Service Bus: Making Service-oriented Architecture Real. IBM Systems Journal 44(4), 781–797 (2005)
17. Ten-Hove, R., Walker, P.: Java Business Integration (JBI) 1.0. Java Specification Request 208 (2005)
18. van Heck, E., Vervest, P.: Smart Business Networks: How the Network Wins. Communications of the ACM 50(6), 28–37 (2007)
19. W3C. SKOS Simple Knowledge Organization System Reference. W3C Recommendation (2009)
20. Wu, B., Liu, S., Wu, L.: Dynamic Reliable Service Routing in Enterprise Service Bus. In: Proc. of the 2008 IEEE Asia-Pacific Services Computing Conference (APSCC 2008), pp. 349–354 (2008)