Implementing a semantic service provision platform

Concepts and Experiences

Die Autoren

Dominik Kuropka, Mathias Weske

Dr. Dominik Kuropka, Prof. Dr. Mathias Weske
Universität Potsdam
Hasso Plattner Institut
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam
Deutschland
0331 / 5509 - 193
{ dominik.kuropka | mathias.weske } @hpi.uni-potsdam.de
http://bpt.hpi.uni-potsdam.de/

Implementing a semantic service provision platform

Concepts and Experiences

Kernpunkte

- Semantik-basierte und automatisierte Suche, Komposition und Ausführung von Diensten ist für ausgewählte Szenarien realisierbar.
- Neue Ansätze zur Fehlerbehandlung basierend auf einer Neubindung und einer Re-Komposition von Diensten zur Laufzeit können durch semantik-basierte Dienst-Plattformen implementiert werden. Somit ist es möglich die Robustheit und die Flexibilität von Dienst-orientierten Anwendungen zu erhöhen.

Stichworte: Dienst-orientierte Architekturen, Semantische Dienste, Dienstsuche, Dienstkomposition, Adaptivität, Flexibilität

Zusammenfassung (nur für das WWW)

Die nahtlose Integration von Informationssystemen spielt eine wichtige Rolle bei der Entwicklung und Wartung von Softwareprodukten. In den heutigen, dynamischen Märkten ist Veränderung eher die Regel als die Ausnahme. Daher ist die Möglichkeit Softwareprodukte effizient an Veränderungen der Anwendungslandschaft anzupassen ein wichtiger Vorteil für eine Unternehmung. Die Vision dienstorientierter Ansätze geht dahin, dass unternehmensrelevante Funktionalitäten von existierenden Softwaresystemen als Dienste bereitgestellt werden, die dann durch eine Komposition der Dienste zu weiteren oder neuen Anwendungen zusammengestellt werden. Leider ist diese Vision bis heute noch nicht in einem hinreichenden Maß in der Praxis umgesetzt worden. In diesem Artikel präsentieren wir eine Übersicht einer Plattform die eine semantische Suche, Komposition und Ausführung von Diensten unterstützt und die oben genannte Vision somit umsetzt. Die Realisierung der Vision wurde prototypisch für ein begrenztes Anwendungsszenario umgesetzt, um ihre grundsätzliche Implementierbarkeit aufzuzeigen. Die Lektionen, die wir bei der prototypischen Umsetzung gelernt haben sind: Erstens, die Anforderungen an die logischen Reasoner sind hoch, wenn man „echte“ Problemstellungen lösen möchte. Zweitens, die formale Spezifikation „realer“ Dienste ist eine aufwändige Aufgabe und drittens, es ist zurzeit nicht einfach reale Szenarien für eine dynamische Dienstkomposition zu finden, weil das Vertrauen in diese Technologie fehlt, weil die Frage nach der Verantwortung im Fehlerfalle nicht geklärt ist und weil die Investitionen in die semantische Modellierung von Diensten sich nur für hoch-dynamische Umgebungen auszahlen.

Implementing a semantic service provision platform

Concepts and Experiences

Abstract

Seamless integration of information systems plays a key role in the development and maintenance of products. In today's dynamic market environments, change is the rule rather than the exception. Consequently, the ability to change products is an effective way to adapt to a changing information technology landscape which is an important competitive advantage of a successful company. The vision of service-oriented computing is to capture business relevant functionalities of existing software systems as services and use service composition to form composite applications. Unfortunately, this vision has yet to be achieved in practice. We present here a high-level overview of a semantic service discovery, composition, and enactment system that realizes this vision. Rather than addressing a fully fledged industrial strength system, we present a research prototype that realizes this vision in a narrow application domain to show the general feasibility of automatic semantic discovery, composition and flexible enactment of services. The lessons we learned from implementing this prototype are: a) The requirements regarding the features of logical reasoners for the implementation of "real" scenarios are high. b) A formal and exact specification of the semantics of "real world" services is a laborious task. c) it is hard to find adequate scenarios, because people don't trust this technology and they don't like to give the control on business processes to a machine, because questions like "Who is responsible?" arise. Furthermore, the application of automated and flexible service discovery and composition at run-time is only cost-effective if changes and volatilities are frequent in the service landscape.

Keywords: service-oriented architectures, semantic services, service discovery, service composition, adaptivity, flexibility

Core points:

- Semantic enabled search, composition and execution of services can be implemented for chosen scenarios.
- Fault tolerant at run-time behaviour can be implemented by semantic service provision platforms by the use of re-binding and re-composition mechanisms.

1 Introduction

Information systems play an increasingly important role in the realization of products that companies provide to the market. Typically, the functionality of several information systems needs to be combined to realize a particular product. Therefore, the seamless integration of information systems plays a key role in the development and maintenance of products. In today's dynamic market environments, change is the rule rather than the exception. Consequently, the ability to change products in an effective way and to adapt products to a changing information technology landscape is an important competitive advantage of a successful company. Existing approaches for the integration of enterprise applications fall short of providing this ease of change.

Service-oriented computing is today regarded as the main technology to provide well specified business functionality realized by information systems. Services allow effective re-use of existing functionality. The vision is to capture business relevant functionality of existing software systems as services and use service composition to plug services to larger, composite applications. Unfortunately, this vision has yet to be achieved. Today it is generally accepted that syntactic specifications of services are not sufficient to provide the high degree of flexibility for finding and composing services in an efficient manner—semantically rich specifications of services are required. In this paper we present a high-level overview on our efforts to implement an adaptive semantic service provision platform which has been realized in the frame of the EU-funded Adaptive Services Grid project which has been driven by ten industry and eleven scientific partners. The research prototype we will present here, realizes the vision of automated service composition and binding of services at run-time in a narrow application domain. Presenting all details of the implementation would go far beyond the scope of this paper, because of the high number of technologies (e.g. SOA, Web services, Grid) and areas of research which influence this work (artificial intelligence, logic and reasoning, agent based negotiation, software engineering, workflow management, etc.). Rather, we want to give you a high-level overview on the results, the possibilities and limitations of our approach. We will provide literature references to more detailed paperwork where applicable.

The remainder of this paper is organized as follows: Section 2 motivates semantic service provisioning; in Section 3 the research problem is presented. Section 4 gives an overview on a reference architecture for a semantic service provisioning system. Section 5 details the responsibilities of the subsystems involved in the service delivery life-cycle. An application scenario is discussed in Section 6, while the lessons learned from the project implementing the reference architecture and the scenario are presented in section 7. After a presentation

of the most corresponding related work in section 8, a discussion of open issues and concluding remarks completes this paper.

2 Motivation

Increasing pressure of competition in software industry forces software vendors to expand their clientele continuously to achieve cost reduction by measures of scale in software development and maintenance. To acquire customers, enhancing offered functionality and making available the existing functionality is of key importance. These requirements lead to increased complexity in system design and development and hence in a disproportionate increase of costs for software development and maintenance. A technical and in many scenarios successful strategy is the separation of functionality into appropriate, reusable and—if applicable—distributed components that realize the system functionality. (Kuropka and Weske 2006) Known technologies for the implementation of such component based systems are Remote Procedure Calls (RPC) (Tanenbaum 2001), the Common Object Request Broker Architecture (CORBA) (OMG 2004) and Web services (W3C n. d.).

For a long time, the separation of software systems into reusable components has been done by software developers almost exclusively along technical aspects. This is particularly appropriate in case software components belong to a single organization. However the demand for short time to market (e.g., flexible adaptation of whole supply and production chains to short-dated fluctuations of demand) increasingly forces the integration of external information systems with the internal systems of an organization. To meet these integration requirements the classical separation into components regarding purely technical reusability aspects turned out to be too fine granular, too much interdependent and therewith hard to integrate. To get a grip on these requirements software components should offer business functionality in the form of dedicated services tailored on the basis of business aspects.

A service in our context is a well defined functionality that is provided on electronic request. To effectively use services, a service-oriented architecture is required. It defines the roles of the involved participants, in particular service provider, service broker and service requester. An extensive description of those roles is given for example in (Burbeck 2000) and (Alonso et al. 2004, pp. 152–155). The usage of services has two advantages for providers as well as for requesters: reduction of complexity and improved integration. The reduction of complexity results from the fact that service-oriented architectures separate the functionality of the system into encapsulated fragments and make them individually available. The adaptation of systems towards business environments takes advantage from the fact that services are

constructed, encapsulated and documented along business aspects and not mainly along technical aspects.

Services can be implemented using different technologies and standards that allow invocation and description of them. Comparatively widespread are W3C Web services, which are using SOAP (W3C 2003), Web Service Description Language (WSDL) (W3C 2001) and Universal Description, Discovery and Integration (UDDI) (OASIS 2002). Though Web services ease the integration of software systems by their business oriented and modular design, they are not able to solve the basic problem of integration: In general it is not possible to forward the results of a service invocation as the input for the next service directly without conversion. Since services are specified in a syntactic manner, computers are not able to “understand” the processed data. As a result, software systems depend on detailed descriptions of the processing steps for the data transformation in form of programs. The writing of such software is a tedious process that requires professionals who understand the technical aspects as well as the business dimensions of the data and the underlying business processes (Kuropka et al. (2005).

Beside the problem of data transformation, the composition of services to obtain business processes is a challenging task. Composition of services is of importance, because it improves the reusability of services and hence contributes to cost reduction. However qualified employees, who are able to discover services using the process specifications or descriptions of the tasks and correctly combine them considering transformation rules are needed for the composition of services. Because of the fact that modern software systems may offer a huge amount of services and that integration may involve several systems developed by different groups, the discovery of proper services by using simple classification criteria—as for example offered by a UDDI directory—is a complicated endeavour. This among other things is caused by differing terminologies and different interpretations of the facts. Because of the high manual effort that is required for the realization of business processes a continuous adjustment of these processes is possible to only a limited extend. Therefore further actions are required for (Web) services to increase their flexibility and implementation efficiency and realize this goal in future service platforms.

The cause of the manual effort in combining services to obtain business processes is the syntactic nature of the usual service and data schema descriptions. Merely the technical invocation of services and the structure of the corresponding data are described. A formal description of the functionality of a service and the meaning of its data structures are usually missing. For a computer the sometimes available informal descriptions in natural language

are mostly useless. Likewise self-explanatory names for methods and data blocks or attributes are not of great value for a computer. Using syntactic descriptions of data structures the computer may only verify if the data blocks of an input message have the required names at the right place to process them according to its program.

To assist a system to comprehend the functionality and the meaning of services and data structures to a limited extent, an expansion of the so far purely syntactic descriptions of services and definitions of data structures by semantic descriptions is required. One possibility to achieve this goal is to use logical expressions consisting of concepts of an existing and generally accepted ontology. Ontology as a term was originally used in the area of philosophy. Computer science has seized the term ontology, but has adjusted the meaning of this term to its needs (Smith 2002, Zúñiga 2001). The fundamental meaning of the term ontology has been agreed upon in informatics, but detailed questions are still under discussion. Consequently no generally acknowledged definition has prevailed yet. This article is based on the following definition: Ontology is a model of linguistic means of expressions several actors have agreed to and that are used for communication between these actors (Kuropka 2004).

Correspondingly an ontology contains concepts and relations between these concepts and respective identifiers that have a fixed and accepted meaning within a certain environment of communication partners. Occasionally ontologies are called models of technical terms in companies. For the usage of an ontology by a software system a description of the ontology using a formal language is needed. Currently popular languages that are partly still in development are the Web Service Modelling Language (WSML) (de Bruijn 2005) in the European and the Web Ontology Language (OWL) (McGuinness and Harmelen 2004) in the American area. Ontologies can be used to pin down the common understanding, the semantics of words and concepts. If two actors are sharing the same ontology they can directly start to communicate about relevant issues without going through a lengthy negotiation process on common understanding. Ontologies written in formal languages, which can be processed by computers, are a possible solution to the challenges of fully automated service-oriented computing in general and web-services in particular. If partners of a service-oriented market agree on one particular ontology as well as one formal language for the description of service specifications, then we have a semantic service specification. This semantic service specification facilitates an automated mediation of the semantics between software components, which makes the vision of dynamic, service-oriented computing feasible (Kuropka and Weske 2006).

3 Research problem

There is a common understanding that semantics plays a key role if services shall be bound and composed in a dynamic way. Furthermore, there are standardized notations for the description of ontologies in this context like WSML and OWL as well as there are ontologies existing, which foster the description of semantic services themselves like Web Service Modelling Ontology (WSMO) (Roman et al. 2006) and OWL-S (OWL-S Coalition n. d). So you may ask where is the research problem? The answer to this question is: We want to find out how a semantic service provision platform can be used and how it does behave in the practice. The aim of the Adaptive Services Grid project (presented below) is to test if and how such a semantic service provision platform can be applied to real world services and scenarios—even if they are small because of resource and time limitations.

4 Semantic Services Reference Architecture

To provide a better understanding of software architectures for semantic service provisioning, we developed a reference architecture in the Adaptive Services Grid (ASG, http://asg-platform.org) project, funded by the European Commission in the Sixth Framework Programme. This reference architecture serves as a blue print for a prototypical implementation of a semantic services platform.

In this section we will give a brief overview on the architecture and the logical components of the ASG platform. A more detailed description of the interplay between the components in order to provide the requested services is presented in the next section. The architecture consists out of five logical components, Facade, Dynamic Service Composition, Semantic Service Discovery, Adaptive Service Management, and Service Infrastructure, as shown in Figure 1.

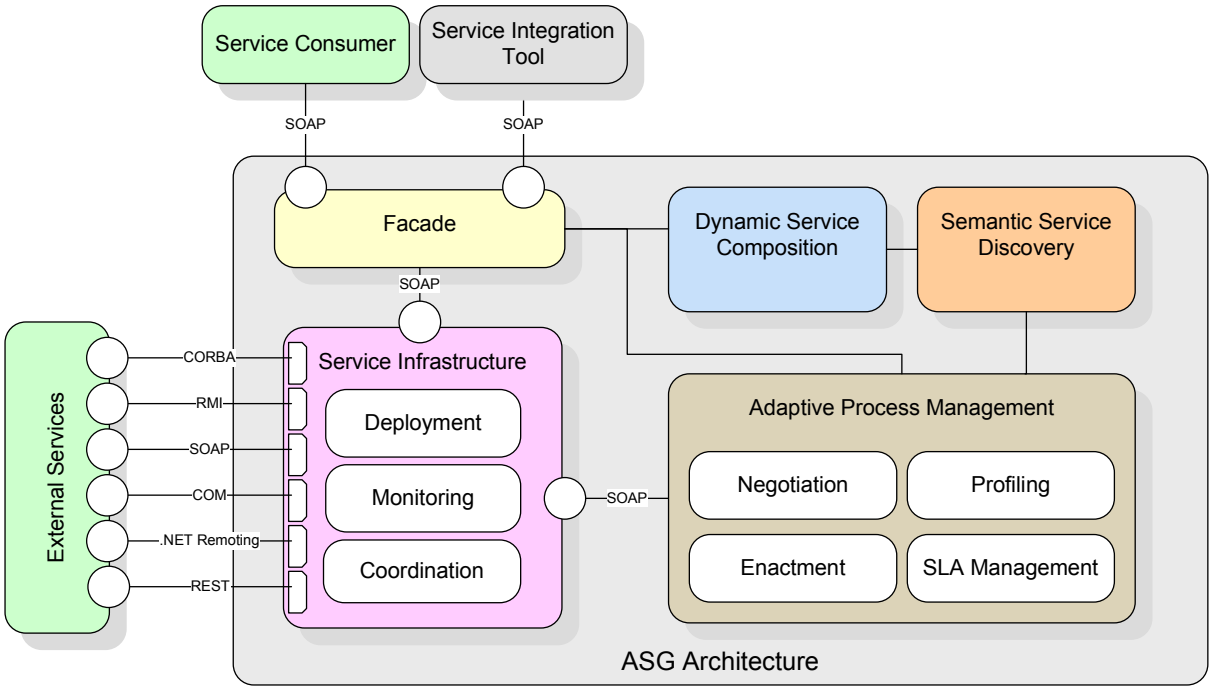

Figure 1: Semantic Services Reference Architecture.

The communication between the external world and the ASG platform is performed via the Facade and the Service Infrastructure component. The Facade component provides programmatic interfaces which can be used by external applications or tools. The domain ontology, which is the base for the description of services and for the semantic service requests, can be accessed and uploaded via the Facade. The domain ontology describes the concepts which are of relevance in the application domain of the ASG platform. These concepts can describe data objects and their relation to each other like for example defining an invoice as a structure consisting out of an address, an order and an amount to be transferred. Furthermore, the ontology can define relationships which are usually not directly represented in usual information systems, but which are useful to describe the functionality of services. For example a relationship “has-cross-at” between maps and coordinates is useful to describe a service which makes a cross on a map at a specific coordinate. With such a relationship the functionality of this service can be formally described by specifying that the effect of the service is that the entered map will be returned having a cross at (“has-cross-at”) the entered coordinates. Finally, the ontology may contain rules like for example: if I know the position of a mobile phone, then I assume that the owner of the mobile phone is currently located at this position.

To be usable, services have to be registered at the ASG platform. This registration happens via an external Service Integration Tool which uses the programmatic interface of the Facade. To properly register a service, following information has to be provided about the ser-

vice: input, preconditions, output, effects, non-functional-properties and the proxy code. All invocations of external services are conducted in the platform via proxies, which are stored and executed in the service infrastructure component. The major task of the proxies is the provision of translations between protocols and mediation of data types. Within the ASG platform all data is transferred via XML (W3C 2006) according to an XML-Schema (W3C 2004) which is derived via rules from the domain ontology. This ensures that data is compatible and can be exchanged between services without manual mediation. In case the service to be registered at the ASG platform is not accessible via SOAP, the proxies are also responsible for a protocol translation. In summary the proxies handle the technical and data format issues of service integration, while the description of input, preconditions, output, and effects are used for the semantic integration. Additionally to the usual input of a service, preconditions can be defined which have to hold for the input. For example it can be defined that an address which is the input of a service, has to be a customer address and not an address of an employee. Similar to this the output defines the output data of a service while the effects can specify additional hints about the meaning and context of the output. For example two different services may exist which take an address as an input and return a phone number as an output. While the first one returns the phone number of the owner of the address, the second returns the phone number of the phone provider who provides the telephone mainlines of the address. In such a case both services have the same input and output even though they provide different functionalities. Such differences can be modelled adequately by preconditions and effects. Finally non-functional-properties can be defined for a service which can be used to specify some quality of service aspects. However these non-functional-properties are only applicable if they are not depending on the concrete input or output data of the service. For example properties like the guaranteed maximal execution duration or the costs per execution for a “Send-SMS” service are applicable as long as both do not depend on the phone number and the text which has to be send. In all other cases the properties are not non-functional and have therefore to be modelled explicitly as input or output data of a service. For more details about non-functional properties and quality of service measures please refer to (Abramowicz et al. 2006).

Service Consumers which usually are end-user applications or application systems use the Facade to pose semantic service requests to the ASG platform. A semantic service request is similar to a service description. However while a service description specifies an existing service a semantic service request specifies a desired service. The ASG platform tries to find a service or a composition of services which is able to meet a posed request. A semantic service request consists out of an initial state, a goal state and optimization criteria. The initial state specifies the data which is given from outside and the context of the data. It is com-

parable with the input and preconditions of a service. The goal state specifies a state which is desired by the consumer. This might include for example some output data and the effects on the world or the data which shall be met after the processing of the request. Optimization criteria can be specified optionally to help the ASG platform to decide in case several services or service compositions are applicable to meet a semantic service request. Semantic service requests are formally specified in WSML, an example for a semantic service request denoted in natural language might look like this: Given the address “Alice; Green Street 1; Greenhill”, the domain name “mydomain.eu” and the following credit card number “1234 5678” as initial state, the following goal shall be achieved: register that domain and charge the payment from the given credit card. Optimization rule to apply: please take the fastest services.

It is worth mentioning that there are no build-in generic end-user interfaces to access the services provided via the platform. The rationale is, that the one hand it is rather hard for a common user to formulate a proper semantic service request without deep understanding of the underlying ontological concepts. On the other hand in most cases some data have to be requested from the end-user (e.g. account id, user name and address) and an automated generation of a user-friendly graphical or web-based interface is not trivial. Therefore semantic service requests are generated by lightweight end-user applications which provide an adequate user interface. These applications take the data from the user and build a semantic service request out of them either by using a template, or by assembling a request out of given building blocks. The basic idea of this approach is similar to today’s usage of relational databases in enterprises; common users do not interact directly via SQL (Date and Darwen 1997) with databases. In fact they use an application which puts their data into an appropriate SQL-template, or which build the SQL query on demand from existing building blocks, as often found in data warehousing applications. While relational databases focus on data management, relieving the programmer from dealing with low-level data storage and consistency issues, the ASG approach takes this idea to a higher level of abstraction: ASG manages functionality provided by services and therefore relieves the programmer from dealing with discovery and composition of services. Additionally to this, ASG raises the reliability by providing failure handling on the basis of service re-binding and re-composition as it will be shown in the next section. The control over this whole service provisioning cycle is monitored and managed by the Facade component.

While the Facade takes requests from the external world, the Service Infrastructure is responsible for the access to existing external services. These services are accessed by the platform via their proxies which are automatically deployed and hosted by the Service Infra-

structure component. As already mentioned, are the proxies responsible for the protocol and data mediation. Furthermore the Service Infrastructure enables the monitoring of services regarding properties like for example the execution time.

In contrast to the Service Infrastructure and the Facade, the components Dynamic Service Composition, Semantic Service Discovery and the Adaptive Process Management are used only internally and have no access to the outer world. The Dynamic Service Composition component is triggered by the Facade to provide services compositions which are able to meet a given semantic service request. The Adaptive Process Management is responsible for the negotiation and binding of services, profiling of the monitored data provided by the Service Infrastructure, the Service Level Agreement Management and finally the most important: the enactment of service compositions by a workflow engine. Finally the Semantic Service Discovery is able to perform a discovery and matchmaking of services. Additionally to this it also provides basic semantic functionalities like a service and ontology repository as well as reasoning functionalities. The functionality of the three components Dynamic Service Composition, Semantic Service Discovery and Adaptive Process Management will be explained in more detail in the next section, since their functionality is in strong relation to the whole service delivery approach of ASG.

5 The Service Delivery Process

The ASG platform performs the delivery of services in a different way to current service provision platforms. In present systems a certain service invocation is usually relayed to a previously statically bound service. This results in a poor utilisation of new services and a poor reliability in case the bound service is down. The semantic service provisioning platform implemented by ASG uses a different approach. Instead of simply binding services to applications at design-time, ASG proposes a sophisticated and adaptive service delivery life-cycle as shown in Figure 2. The entry or initial point of this delivery life-cycle is the already described semantic service request (refer to section 3). In contrast to a static service binding the semantic service request does consist of a description of what shall be achieved and not which concrete service has to be executed. The semantic service request describes the initial and goal state and consists therefore among other things out of the given data, data types and conditions which are met by the data as well as the desired type of data and desired effects beyond. The data types, conditions and effects are all specified in relation to a common set of concepts—the domain ontology.

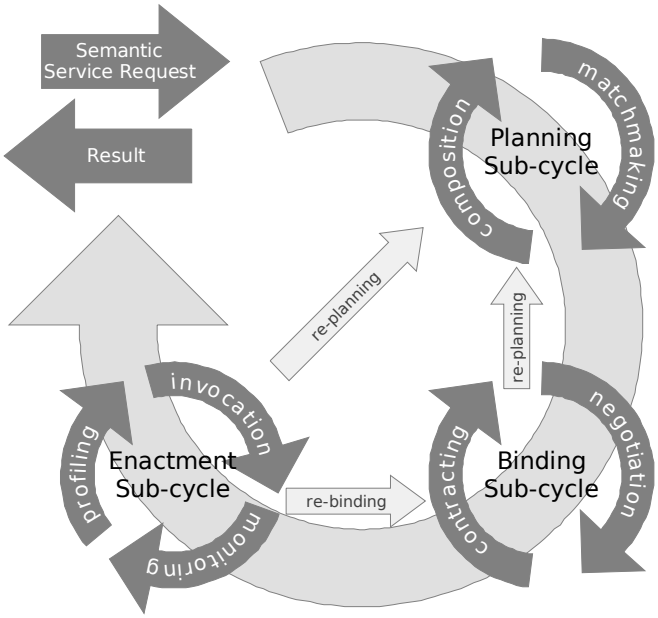

Figure 2: ASG service delivery lifecycle.

5.1 Planning Sub-cycle

The Planning Sub-cycle is the first step in processing of a semantic service request and it is implemented by the Dynamic Service Composition component. This component is able to create an abstract composition of services which is able to meet the semantic service request. Abstract composition means, that the composition does not directly bind existing services. Rather the services are represented by semantic service specifications which act as placeholders for the real services. This proceeding allows a late binding of services and features a better reusability of service compositions which is useful for performance issues. It is for example possible to cache abstract service compositions for frequent semantic service request, so that the need for the expensive planning of service compositions can be reduced. Abstract service compositions are put down in an extended form of Business Process Execution Language (BPEL) (OASIS 2004). Standard BPEL demands activities to be bound to existing services. Therefore an extension is necessary to allow a representation of activities by semantic service specifications as used here.

At the beginning the Dynamic Service Composition component tries to find a service which perfectly matches the semantic service request. Perfectly matching means in this context, that the service is able to process the given data as input, that all pre-conditions for the execution of the service are fulfilled, and that the service output fits to the desired type of data and effects. In case of successful matchmaking the Planning Sub-cycle is completed and it returns an abstract service composition consisting just out of this one service. If the desired functionality can not be provided by the invocation of just one service, the platform tries to

find an abstract composition of services which is able to meet the semantic service request. The composer starts composing from the initial state of the semantic service request. Given this state the composer searches for services which are executable in this state. By taking the results and effects into account the composer searches successively for new executable services until the goal state of the semantic service request is met or a further composition is not possible or reasonable. For further details on the composer and the composition algorithm please refer to (Meyer and Weske 2006). In case of successful composition an abstract service composition, represented in a BPEL document, is created and forwarded to the next sub-cycle. Otherwise the processing of the semantic service request is aborted with an error.

5.2 Binding Sub-cycle

Abstract service compositions are transformed into bound service compositions by the Adaptive Process Management component in the Binding Sub-cycle. This happens by binding the semantic service specifications—which act as placeholders—to concrete services. The Adaptive Process Management component implements two different binding strategies: binding via selection and binding via negotiation.

Binding via selection means that for each activity of the composition the component asks the Semantic Service Discovery for services which are semantically matching that activity. In case several services are returned, then the “best” one of them is selected according its non-functional-properties and the optimization criteria of the semantic service request. It is worth mentioning that the Adaptive Process Management component also supports a long-term profiling of services. This means that the component is able to use monitoring data which is provided by the Service Infrastructure to observe the long-term behaviour of a service. For example with this, it is possible to get a statistics of the average execution time of a service, or the percentage of violations of the non-functional properties like exceeding of a specified maximal duration of a service, or the percentage of failed requests due to the service being down. All these profiled data can also be used to optimize the selection of services.

Binding via negotiation is an extended form of binding which works like this: For each placeholder the Adaptive Process Management component starts a negotiation on negotiable service properties with all matching services. The platform tries to find a combination of services for all activities, which fits as good as possible to the optimization criteria of the semantic service request. Negotiation builds up on the idea that non-functional properties of a service are not static but can be negotiated dynamic at run-time. For example it might be useful to specify the maximal duration of a service execution or the costs in relation to the load on the

service infrastructure providing a particular service. When an agreement with a particular service is achieved a digital contract it is set up and digitally signed by both parties. Further details on this topic are presented in (Momotko et al. 2006). In ASG services do not necessarily need to support negotiation mechanisms. In such cases only a binding via selection is viable. The result of the Binding Sub-Cycle is a usual BPEL document in which all activities are bound to concrete services.

5.3 Enactment Sub-cycle

The third step in processing of semantic service requests is the Enactment Sub-cycle. It receives bound service compositions from the Binding Sub-cycle coded as a BPEL document and enacts them. Two components are involved in this: Adaptive Process Management and Service Infrastructure. The Adaptive Process Management component provides a workflow engine which is responsible for the enactment of the service composition activities as a workflow. In the project we have used the commercial Rodan ObjectOffice Workflow™ Engine for this task. This workflow engine invokes all activities of the service composition successively or in parallel, where possible.

The Service Infrastructure comes into play when an activity has to be executed. Since every activity is bound to a service, a proper service call has to be generated. This generation of a call is handled via a service proxy that has been stored during the registration of the service. In this context, the Service Infrastructure is responsible for the deployment and execution of the service proxies. For the execution of an activity the workflow engine requests a handle of the proxy representing the service bound to the activity. In the next step, the proxy is called via the handle and all input data is send to the proxy. The proxy takes care of a proper data type and protocol transformation and calls the real service. In case the service returns some data, this data is transformed in a way that it is compatible to the data schema specified by the domain ontology. By receiving the data from the proxy, the workflow engine can successfully close the activity and continue. After successful enactment of the whole service composition the final result is collected and it is send back to requester.

5.4 Failure handling by re-binding and re-planning

The ASG service delivery life-cycle includes two mechanisms to handle dynamics when it comes to explicitly considered or unconsidered failures of services: re-binding and re-planning. The definition of the term failure used here is compliant to the definition in (Avinzienis et al. 2001).

Considered failures are well known failures which might occur during the execution of a service and are therefore explicitly specified as possible (even though not desired and therefore hopefully seldom) results of a service. One example for such a considered failure is the rejection of credit card by a credit card withdrawal service, which might occur if the credit card data is invalid or expired. Another example is the loss of a package by a package shipping service. Considered failures are handled in ASG by conducting re-planning. In case a considered failure occurs during the invocation of a service, the Planning Sub-cycle is triggered to find a new composition. Referencing to the package shipping service example, this might mean that a new package is seized and send. Naturally not all considered failures can be handled by re-composition. In the case of the credit card withdrawal service example, a recovery is not possible and it is even not desired if a credit card is turned out as being invalid.

In contrast to the considered, are the unconsidered failures not explicitly specified in a service specification. Unconsidered failures are usually low-level issues like network failures which are raised pre or during the invocation of a service. The platform has no detailed information about semantic effects of such failures except that these failures just happen at a given point in time. For this reason it assumes in such cases of failure, that the according service simply has not been executed and thus its desired results and effects are not achieved. Such unconsidered failures are handled in up to two phases. In the first phase the platform tries to recover the failure by re-binding. This triggers a new pass of the Binding Sub-cycle. A search for an alternative equivalent service to the already invoked and failed service is conducted by negotiation with or selection of proper services. If the search is successful, the new service is invoked as substitution for the old one. Else the second phase is conducted. In this second phase the ASG platform tries to recover the unconsidered failure by re-planning the composition in the Planning Sub-cycle. In case this planning is successful the Binding Sub-cycle is invoked. After the outstanding service placeholders have been bound to concrete services the new composition is enacted in the Enactment Sub-cycle.

It is worth to mention, that re-binding and re-planning are similar to the original binding and planning tasks. Like binding, also re-binding is provided by the Adaptive Process Management component. To conduct a proper re-binding the component is informed by the Service Infrastructure about the failed service. Therefore the Adaptive Process Management component can take care that this service is not selected or negotiated again during the re-binding phase. Additionally to this, this kind of information is useful for a profiling of the service. For example this information can be used to avoid the usage of the service in upcoming service binding tasks in the near feature. By this way it can be avoided that next requests run into the same situation of failure due to the service being not available for a shorter period of time.

The major part of re-planning is handled by the Dynamic Service Composition in the same way as usual planning. However the Adaptive Process Management is responsible to extract the information about the current state of the enactment of the service composition, which is the basis for the re-planning. In the case of re-planning the initial state of the semantic service request has to be replaced by the state of the service composition at the time of failure. This is necessary to avoid repeating requests for services which already have been successfully executed prior the failure. With the current state at time of failure and the old goal from the semantic service request the Dynamic Service Composition component is able to find alternative service compositions in case some proper services are available which together can replace the functionality of the failed service.

6 Application Scenario

Next to others, the following small scenario has been successfully implemented by the ASG project on the basis of existing 'real-world' services and the ASG platform reference implementation. It is an example based on a business-to-business (B2B) wholesale model of an Internet Service Provider (ISP) who is a member of the ASG project. The ISP is specialised on products like domain registration and web hosting. The reason to choose this scenario was, that it is a real scenario of one of our project partners and that there are some digital, public, real-world services available to implement this scenario. The ISP uses the ASG platform in a role as a hosting provider to bundle and integrate external services provided by for example registrars like Denic, VeriSign and Directi and by digital payment providers like SaferPay or PayPal (refer to Figure 3). The integrated services are provided by the ISP to Resellers who use the services either internally or offer them to their customers. One benefit for the resellers is that they do not need to deal with the individual integration of the various basic services, since it is already provided by the ASG platform. The resellers access the various services via semantic service requests which are then handled by the service provisioning lifecycle as shown in Figure 2 and discussed in section 4.

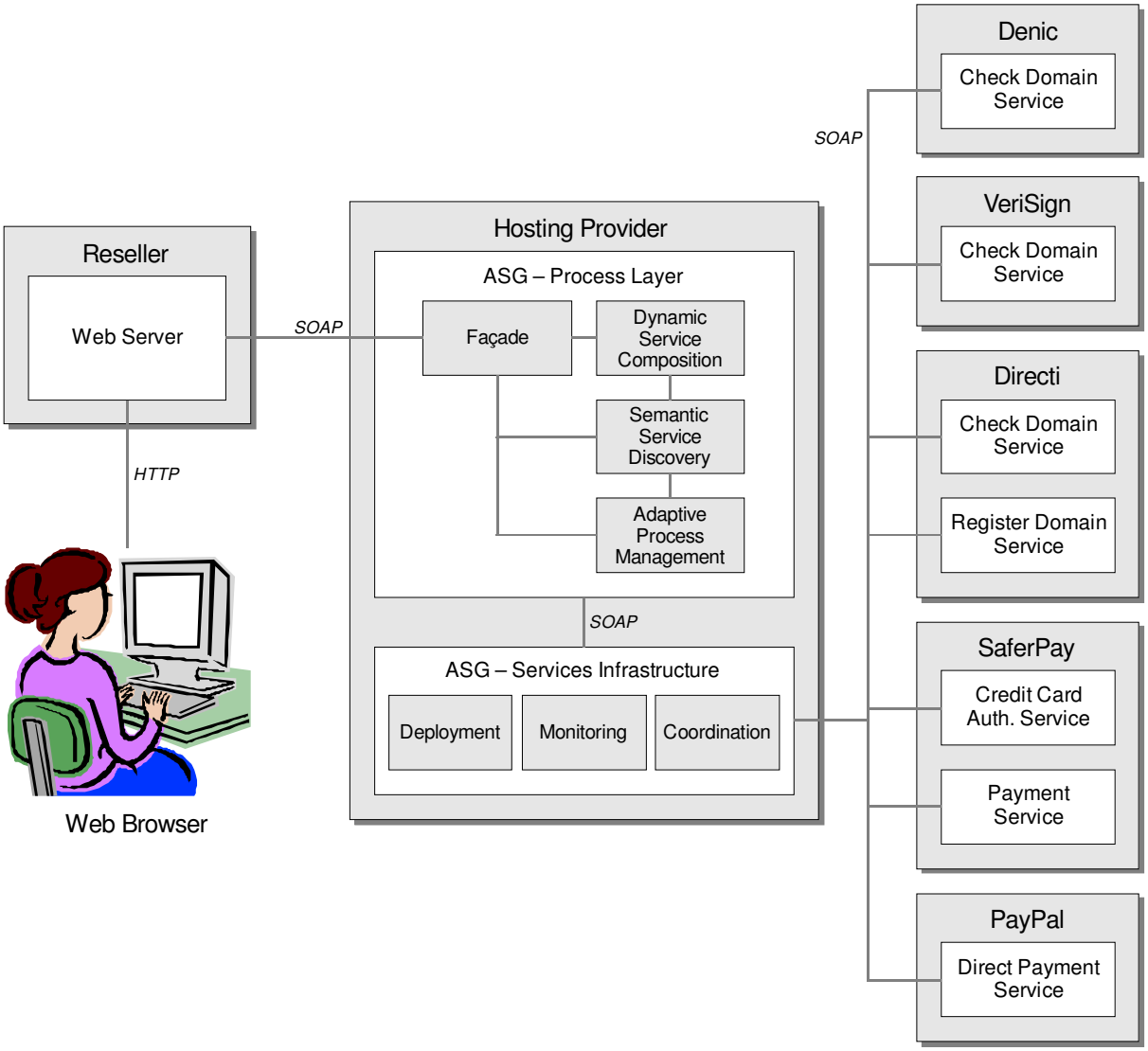

Figure 3: Scenario architecture.

Another benefit the reseller has from using the ASG platform is the automated failure handling. As already described in section 4, the ASG platform supports the handling of failures by re-binding and re-planning. Figure 4 shows an already bound service composition which is automatically created by the ASG platform and how re-binding can affect such a composition. Translated to natural language, the semantic service requests which is the origin of the composition looks like this: “Given the following domain *x* and the following user data *y* the goal state to achieve is that the domain is registered at an registrar, our name server is updated, a web hosting account is created, and a default forwarding for the domain is set up.” Both, the (directi and verisign) register domain services and the plesk[1] create domain service expect as precondition that the existence of a domain is checked prior their execution. There-

[1] Plesk is a commercial control panel solution for hosting providers. The control panel is designed to simplify the management and administration of web sites. Plesk control panel automates a large number of tasks often recurring in the context of web hosting.

fore, a check domain service is executed at the beginning of the service composition in Figure 4. The plesk create domain service can naturally only be executed on existing web hosting accounts, therefore a web hosting account is created before a domain is created in plesk. Additonally to this, Figure 4 shows how the service composition is recovered in case the denic check domain service fails, for example because of network problems. In this case the platform searches for a semantically equivalent service like the verisign check domain service and replaces (re-binds) the failed service. In our scenario also the directi register domain service can be replaced by a semantically equivalent verisign register domain service.

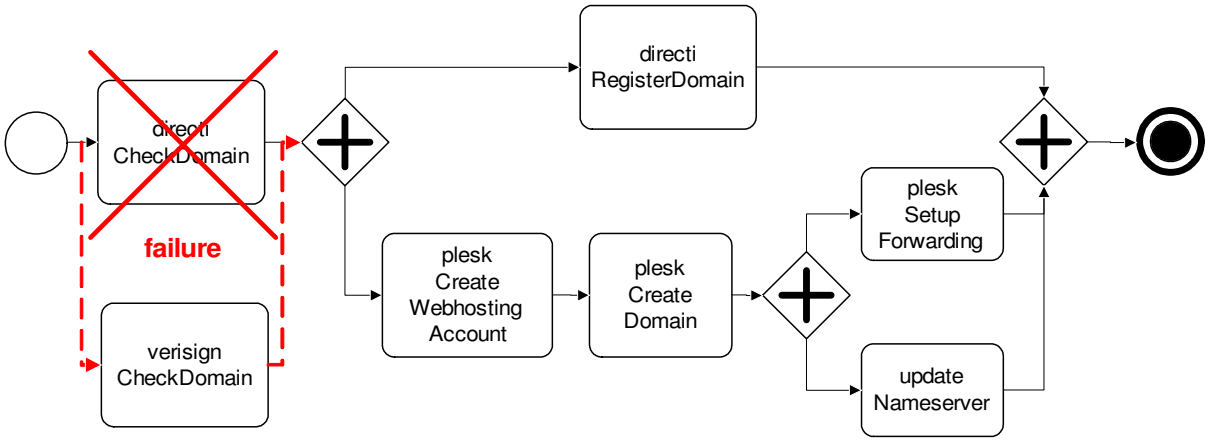

Figure 4: Service re-binding sample.

Figure 5 shows how re-planning can affect a service composition. In our scenario we have implemented two providers for handling of payment activities via credit card. We where able to use either the payPal payment service which can handle the payment activities in one step or the saferPay payment service which needs two steps for this task. Both alternatives are semantically equivalent and can be therefore replaced by each other. In such cases the ASG platform is able to handle a failure of for example the payPal service by re-composing a service composition and replacing it by the two-step saferPay service.

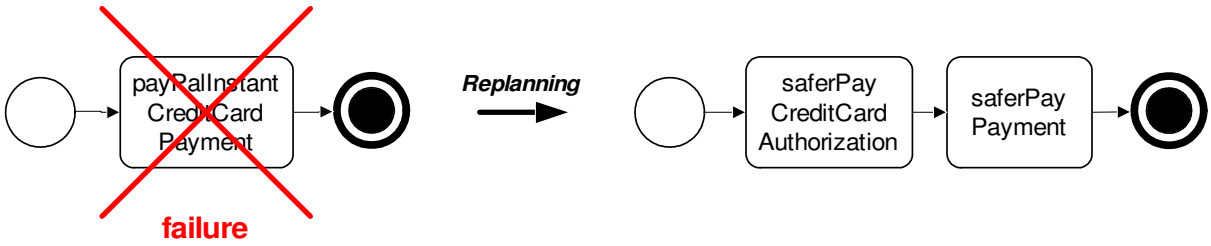

Figure 5: Service re-planning sample.

7 Lessons learned

The first lesson which we learned at the beginning of the project was: Our requirements regarding the features of logical reasoners exceeded by far what was available on the market, and especially on the open source market at the beginning of the project in 2004. From our point of view a fulfilment of the following criteria for a reasoner was desired: We needed a matured reasoning component which is easy to integrate and which supports multi-user access with a transaction-based isolation. Furthermore a persistent storage of the knowledge base was required. Additionally to this, the reasoner should support reification (since we use a reification-based matchmaking algorithm as described in detail in (Kuropka and Meyer 2005)), open-world assumption, and a powerful query language.

A second lesson we learned was, that a formal and exact specification of the semantics of "real world" services is a laborious task. To support an automated composition or even just a flawless automated discovery and binding of services at run-time, services and their preconditions, effects etc. have to be specified very precisely. On the one side logical languages, which are known to be not easy to learn and handle are a significant hurdle. On the other side, the need for formal correctness of the ontologies does not leave any space for fuzziness, which makes the agreement on one domain ontology a tough task. Furthermore, both the service developers and requesters need deep knowledge regarding the formal modelling languages and the ontologies used. Service developers need this knowledge to describe the functionality of their services precisely, while service requesters need this knowledge to describe what kind of functionality they are looking for. It is important to note, that this issue can not be simply mastered by providing better (e.g. graphical) ontology, service description, or query modelling tools, since it is not just a problem of presentation but it is a problem of domain and modelling complexity. However, mature modelling tools might be of significant help to reduce the hurdles a bit.

Finally, we experienced that it is hard to find adequate scenarios on which an adaptive service provision platform can be applied at the current state. There are various reasons for this observation: People don't trust this technology (at the moment) and they don't like to give the control on business processes to a machine, because questions like "Who is responsible?" arise. Furthermore, in some cases parts of a process are pre-specified by laws or regulations. Another aspect is the fact, that some scenarios simply exceeded the modelling, reasoning or composition possibilities of our prototype. Furthermore, the modelling effort of some scenarios was too high to provide them in-time. Since the creation of ontologies and the modelling of semantic services and semantic requests is such a laborious task, the economic efficiency of this task has to be evaluated in detail. The application of automated and

flexible service discovery and composition at run-time is only cost-effective if changes and volatilities are frequent in the service landscape, so that the cost-reduction by the automation of service composition etc. exceeds the investments of the modelling and maintaining the ontology, semantic service specifications and requests.

8 Related Work

The Web Service Execution Environment (WSMX) is a semantically-enabled service-oriented architecture that enables the creation and execution of semantic Web services based on the Web Service Modelling Ontology (WSMO). Providers can use WSMX to register and offer their services, while requesters can use it to dynamically discover, mediate and invoke Web services. In contrast to ASG, current WSMX does not support automated service composition, but it implements late binding as a strategy for adaptation, similar to ASG. It uses its own matchmaking component for the discovery of appropriate services. Additionally to data mediation, WSMX supports also process mediation which is not directly addressed by ASG. (Haller et al. 2005, Vitvar et al. 2007).

Another WSMO-based approach is the INFRAWEBS Integration Framework (Agre and Marinova 2007). The current version of the INFRAWEBS Framework assumes that ontologies for descriptions and data types of the services are compatible to each other. However it is planned to use mediators in the future as soon as they get mature enough to be used. This is a different approach to ASG, since ASG assumes that services are described on the base of a common ontology, while mismatching data types are resolved in ASG by data mediation which is conducted by the service proxies. Similar to ASG, INFRAWEBS allows the usage of templates for goals (semantic service requests in ASG), to ease the application of the platform for non-expert users. In contrast to ASG, the goals in INFRAWEBS have to be either atomic and thus have to meet exactly some existing service descriptions or are composed out of dedicated sub-goals. Therefore the capability of the composer in INFRAWEBS is limited by the hand made composition of a goal into its sub-goals. This allows late binding and re-binding in case of failures, while a free composition and re-composition of services is not possible at run-time.

A prominent example for adaptivity in service-oriented architectures is the Meteor-S project. Meteor-S is a framework for semantic web services supporting semantic description, discovery and composition of services. It allows the manual composition of services by the use of semantic operation templates which are similar to abstract compositions in ASG. These templates describe the functionality of an operation at each step of the composition. Meteor-S supports a late binding mechanism to discover and bind proper services at run-time for each

operation of the composition. Non-functional service properties and composition constraints are taken into account by the implemented discovery and binding mechanisms. Furthermore, the framework supports a re-binding of individual services at run-time to master service failures. However, an automated composition of services and negotiation of non-functional properties is not supported (Verma et al. 2005).

9 Conclusion

This paper presented a high-level picture of semantic service discovery, composition and flexible enactment of services. By restricting the discussion to a narrow domain, a reference architecture with subsystems and their responsibilities has been presented that effectively realizes the vision of automatic service composition and enactment. The flexible enactment engine can cope with flexible service landscapes in which services emerge and disappear, so that particular properties of service-oriented architectures are well taken care of. The results presented in this paper are based on a detailed domain ontology. In real-world settings, the development of domain ontologies might incur massive effort, so that the commercial gain provided by the flexibility might or might not rectify the efforts in all scenarios. Furthermore security and performance issues haven't been addressed in the prototype presented as far as it would be needed for a productive system. This is up to future research.

A demonstrator of the semantic service provision prototype developed by the Adaptive Services Grid can be downloaded from the project homepage at http://asg-platform.org. Furthermore, major parts of the prototype are published under an open source licence and are available for download, too.

Acknowledgement: This article reports on work that has been done in the context of the Adaptive Services Grid project (contract number: 004617, identifier: FP6-2003-IST-2). The authors are grateful to the ASG project members and appreciate their valuable work in the project.

References

Agre, G.; Marinova, Z. (2007): An INFRAWEBS Approach to Dynamic Composition of Semantic Web Services. In: Cybernetics and Information Technologies (CIT), Volume 7, No. 1, ISSN 1311-9702, Bulgarian Academy of Sciences.

Abramowicz, W., Kaczmarek, M., Zyskowski, D. (2006): Duality in Web Services Reliability. In: AICT 2006, ICIW 2006, Advanced International Conference on Telecommunications 2006, International Conference on Internet and Web Applications and Services, Guadeloupe, French Caribbean, ISBN 0-7695-2522-9.

Alonso, G.; Casati, F.; Kuno, H.; Machiraju, V. (2004): Web Services: Concepts, Architectures and Applications. Springer, Berlin.

Avizienis, A. ; Laprie, J. ; Randell, B. (2001): Fundamental Concepts of Dependability. UCLA \& LAAS \& Newcastle University. http://citeseer.ist.psu.edu/avizienis00fundamental.html downloaded 2007-10-22

de Bruijn, Jos (2005): The Web Service Modeling Language WSML. http://www.wsmo.org/TR/d16/d16.1/v0.21/ downloaded 2007-10-22

Burbeck, Steve (2000): The Tao of e-business services—The evolution of Web applications into service-oriented components with Web services, IBM Software Group.

Date, C.J.; Darwen, H. (1997): A Guide to the SQL Standard. 4th Edition, Reading, MA, Addison-Wesley.

Haller, A.; Cimpian, E.; Mocan, A.; Oren, E.; Bussler, C. (2005): WSMX — a semantic service-oriented architecture. In: Proceedings of the International Conference on Web Service (ICWS 2005), Orlando, Florida.

Kuropka, D.; Bog, A.; Weske, M. (2005): Semantic Enterprise Services Platform: Motivation, Potential, Functionality and Application Scenarios. In: Proceedings of the thenth IEEE international EDOC Enterprise Computing Conference. Hong Kong, pp. 253 – 261. ISBN 0-7695-2558-X, ISSN 1541-7719.

Kuropka, D; Meyer, M. (2005): Survey on Service Composition Technical Report of the Hasso-Plattner-Institute, 10, ISBN 3-937786-78-3. http://kuropka.net/files/HPI_10_Serv-Comp-Survey.pdf downloaded 2007-10-22

Kuropka, D. (2004): Modelle zur Repräsentation natürlichsprachlicher Dokumente – Information-Filtering und -Retrieval mit relationalen Datenbanken. In Series: Advances in Information Systems and Management Science, 10. Editon. Logos Verlag, Berlin.

Kuropka, D.; Weske, M. (2006): Software Development Dynamics: Current Trends and Future Developments. In: Welfens, P.; Weske, M.: Digital Economic Dynamics. Springer Berlin, pp. 5 – 24.

McGuinness, D; van Harmelen, F. (2004): OWL Web Ontology Language Overview. Web Ontology Working Group at the World Wide Web Consortium (W3C). http://www.w3.org/TR/owl-features/ downloaded 2007-10-22

Meyer, H.; Weske, M. (2006): Automated Service Composition using Heuristic Search. In: Proceedings of the Fourth International Conference on Business Process Management (BPM 2006), Vienna, Austria. http://bpt.hpi.uni-potsdam.de/twiki/pub/Public/HaraldMeyer/asc.pdf downloaded 2007-10-22

Momotko, M.; Gajewski, M.; Ludwig, A.; Kowalczyk, R.; Kowalkiewicz, M.; Zhang, J. Y. (2006): Towards Adaptive Management of QoS-aware Service Compositions — Functional Architecture. Service Oriented Computing Conference (ICSOC).

OASIS (2002): UDDI Version 2 Specifications. http://www.oasis-open.org/specs/index.php#uddiv2 downloaded 2007-10-22

OASIS (2004). Web Services Business Process Execution Language (WS-BPEL). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel downloaded 2007-10-22

OMG (2004): Common Object Request Broker Architecture: Core Specification. Version 3.03. http://www.omg.org/docs/formal/04-03-01.pdf downloaded 2007-10-22

OWL-S Coalition (n. d.): OWL-S 1.1 Release. http://www.daml.org/services/owl-s/1.1 downloaded 2007-10-22

Roman, D; Lausen, H; Keller, U (2006): D2v1.3. Web Service Modeling Ontology (WSMO) — Final Draft. http://www.wsmo.org/TR/d2/v1.3/20061021 http://www.wsmo.org/TR/d2/v1.3/20061021

Smith, B. (2002): Ontology and Information Systems. http://wings.buffalo.edu/philosophy/faculty/smith/ articles/ontologies.htm (not available on the server anymore)

Tanenbaum, A. (2001): Modern Operating Systems. 2nd Edition, Prentice Hall.

Vitvar, T; Mocan, A; Kerrigan, M; Zaremba, M; Zaremba, M; Moran, M; Cimpian, E; Haselwanter, T; Fensel, D. (2007): Semantically-enabled Service Oriented Architecture: Concepts, Technology and Application. In: Service Oriented Computing and Applications.

Verma, K.; Gomadam, K.; Sheth, A.P.; Miller, J.A.; Wu, Z. (2005): The METEOR-S approach for configuring and executing dynamic web processes. Technical report, LSDIS LAB, University of Georgia, Athens, Georgia.

W3C (n. d.): Web Services Activity. http://www.w3.org/2002/ws downloaded 2007-10-22

W3C (2001): Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl.html downloaded 2007-10-22

W3C (2003): SOAP Version 1.2 Part 0: Primer. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/ downloaded 2007-10-22

W3C (2004): XML Schema Part 0: Primer Second Edition—W3C Recommendation 28 October 2004. http://www.w3.org/TR/xmlschema-0/ downloaded 2007-10-22

W3C (2006). Extensible Markup Language (XML). http://www.w3.org/XML/ downloaded 2007-10-22

Zúñiga, G. (2001): Ontology: Its Transformation From Philosophy to Information Systems. In Proceedings of Formal Ontology in Information Systems, pp. 187–197.