Towards a service composition and enactment platform

Dominik Kuropka* and Mathias Weske

Business Process Technology,
Hasso Plattner Institute for IT Systems Engineering,
University of Potsdam,
Prof.-Dr.-Helmert-Straße 2-3, Potsdam 14482, Germany
Fax: +49-331-5509-189
E-mail: dominik.kuropka@hpi.uni-potsdam.de
E-mail: mathias.weske@hpi.uni-potsdam.de
*Corresponding author

Abstract: Service-oriented software architectures that are aware of business processes will form the core of operational IT landscapes in the future. This contribution starts with a brief introduction of the state of the art in service-oriented architectures. The authors argue that semantically rich descriptions of services are essential to tap the full potential of service-oriented architectures in enterprise environments. This regards matchmaking and binding of services, integration of new services as well as the cost-efficient development of added value services by composing semantically described basic services. This paper introduces a semantic service platform that implements dynamic matchmaking, composition and binding of semantically described services. Finally its functionality and a possible application scenario are outlined.

Keywords: service-oriented architectures; service platforms; semantic services; service composition.


Biographical notes: Dominik Kuropka is a Senior Research Assistant at the business process technology research group at the Hasso Plattner Institute. He was the Scientific Coordinator of Adaptive Services Grid, an Integrated Project in the Sixth Framework Programme of the EU. His research interests include ontology-supported information filtering and retrieval, semantic Web services and services-based software architectures. He has published 15 scientific papers in journals and conferences. He is a Member of GI and a Member of the EMISA executive committee.

Mathias Weske is Chair of the business process technology research group at Hasso Plattner Institute for IT Systems Engineering at University of Potsdam, Germany. His research interests include business process management, service-oriented computing, semantic services and process-oriented software product lines. He has published eleven books and over 40 scientific papers in journals and conferences. He is a Member of ACM, IEEE and GI and he is the Chair of EMISA, the German Computer Science Society Special Interest Group on Development Methods for Information Systems and their Application.

1 Introduction

Today’s software architectures in business environments are challenged to meet the continuously changing requirements of the market in a cost-efficient and timely manner. Therefore, the proper design of software architectures regarding flexibility and adaptability has moved into the focus of business and scientific communities. A central requirement of these architectures is the delivery of clearly defined functionality that is available in a modular design and can be used in a flexible way. Service-oriented software architectures are a promising approach to meet the requirements of a continuously changing market.

It is assumed that they will form the core of operational IT landscapes in the upcoming future.

In this contribution, central requirements are identified that are not satisfied by service-oriented architectures so far. The authors argue that semantically rich descriptions of services are essential to tap the full potential of service-oriented architectures. This regards dynamic matchmaking and binding of services, automatic integration of new services as well as the cost-efficient development of added value services by composing semantically described basic services. Finally, this contribution introduces a semantic service platform, developed in the Adaptive Services Grid (ASG) project. The platform
provides dynamic matchmaking, composition and binding of semantically described services. Its implemented functionality, and in particular, the life cycle of service delivery are discussed here. Additionally, possible application scenarios will be outlined.

2 State of the art

Increasing pressure of competition in the 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 realise 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) (Object Management Group, 2004) and Web services (World Wide Web Consortium, 2002).

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 organisation. 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 organisation. For 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.

In the context of this paper, a service 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 Web services, which are using SOAP (World Wide Web Consortium, 2003), Web Service Description Language (WSDL) (World Wide Web Consortium, 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 one 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.

A concrete yet simple integration example for such a problem is shown below: we assume that an enterprise wants to integrate a customer management system of an Anglo-Saxon manufacturer into the existing enterprise resource planning system of a company. Functionalities of the customer management system shall be integrated into the following customer request handling process: in case an official in charge is called, the most likely address of the calling customer and all the open orders of this customer or customer group shall be displayed to the official automatically by using the transmitted telephone number for identification. For the reason of simplicity, it is assumed that this scenario can be implemented by simply invoking two services. The first service is provided by the customer management system and offers the functionality to determine an address on the basis of a telephone number. Since this is an English software provider, the specified data format for the address as the output of the service is: ‘full_name’, ‘address’, ‘city’, ‘state’, ‘ZIP’, ‘country’.

The second service needed for this process is provided by the existing enterprise resource planning system, it uses the address as an input and offers the functionality to open a screen form on the desktop of the official that displays a choice of all open orders for the given address. Since this service is provided by a German system in our scenario, the address as an input for the service is assumed to correspond to the following data format: ‘Name’, ‘Straße’, ‘Hausnummer’, ‘PLZ’, ‘Stadt’, ‘Land’. Each block of the address contains the following parts of the information: name, street, house number, post code, town and country. It is easy to see that the two data formats do not match. Current software systems are not able to convert the information from the first data format to the second data format without the aid of a programmer who specifies corresponding transformation rules. Even in this small example, the implementation of such transformation rules is relatively laborious. As an example, the street and the house number have to be extracted from the block ‘address’ using appropriate rules. A simple one-to-one transformation
that merely exchanges the block names is not applicable especially to complex data structures, as for example, invoices or orders. Additionally, in the worst case, such transformation rules might be needed for every composition of two services.

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 natural language 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 realisation of business processes, a continuous adjustment of these processes is possible only to a limited extent. Therefore, further actions are required for (Web) services to increase their flexibility and implementation efficiency and realise this goal in future service platforms.

3 Semantic description of services

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 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. It is of no difference for a computer if data blocks are labelled with self-explanatory names like ‘name’, ‘street’, ‘house_number’, ‘zip_code’, ‘town’ and ‘country’ or if they are assigned abstract names like ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’. 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 software 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 paper 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 Europe and the Web Ontology Language (OWL) (McGuinness and van Harmelen, 2004) in the American area.

4 A semantic services platform

In this section, the main concepts of a semantic services provisioning platform are introduced. We have implemented a prototypical version of such a platform in the EU-funded (ASG, http://asg-platform.org) project. Our platform discovers and composes services using semantically specified data structures and services, based on domain ontologies using WSML. An unambiguous, automated transformation rule facilitates the transformation of the ontology to a specific XML schema, used to capture data exchange between services.

To illustrate domain ontologies, Figure 1 shows a simplified visualisation of a section of a domain ontology referring to the above mentioned integration example. Furthermore, the figure shows how the data structures of the ontology are represented as an XML schema and how the data structures of the above mentioned sample are mapped to the schema of the platform.

The number of required mappings of schemes depends on the number of systems to integrate and it is of proportional...
growth, which is a benefit of using a central data schema that is derived from the domain ontology. If such a central data schema is not used, the number of mappings grows in a fast disproportionate manner with the number of systems to integrate (Kuropka et al., 2006).

A service needs to be registered to become applicable within the platform. At registration time, the following information about a service will be stored: the semantic service specification, non-functional properties of the service and the grounding of the service. The semantic service specification defines input and output data structures referring to the domain ontology. Preconditions and effects (exceeding simple data input and output descriptions) may be defined in the semantic service specification. This allows to specify a service to have a telephone number as input and an address as output data, and that the returned address is not just any address but the address for the specified telephone number. (The example assumes that each telephone number is associated with exactly one address). This information is required for a complete semantic specification of the service, otherwise the relation between input and output data in the example above is imprecise. Another service might exist, which also has a telephone number as input and an address as output data that returns the address of the telephone provider for the specified telephone number instead. Without a complete semantic specification of a service including preconditions and effects, such a distinction of functionality is only possible by the means of natural language documentation, which cannot be automatically processed.

Figure 2 depicts a simplified visualisation of the semantic specifications of the services from the integration example (services 1 and 2). It is worth mentioning that service 3 is of the same syntactical type as service 1 regarding input and output data, but instead of returning a customer’s address, it returns the address of the telephone provider supplying the specified phone number. Such a difference of functionality is not visible on the level of purely syntactical definitions of a service, but can be represented and distinguished by using formal semantic specifications.

5 Service delivery

The processing of a call and the delivery of services by the ASG platform differs from approaches of current service provisioning platforms. In present systems, a certain service invocation is always 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 the ASG project 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 3. This life cycle has three basic steps: planning, binding and enactment. The entry or initial point of this delivery life cycle is a semantic service request. In contrast to a static service binding the semantic service request, it 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.

The Planning Subcycle is the first step in the processing of the semantic service request. At the beginning, the ASG platform 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 preconditions 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 Subcycle is completed. Otherwise, the platform tries to find an abstract composition of services which is able to meet the semantic service request. Abstract composition
When an agreement with a particular service is achieved, the 
Enactment Subcycle (Momotko et al., 2006). A digital contract is set up and signed by both parties.

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 (e.g. by buffering compositions). The activity of creating a service composition starts from the initial state of the service request. Given this state, services are searched which are executable in this state. By taking their results and effects into account, new executable services are successively searched until either the goal state of the semantic service request is met or a further composition is not possible or reasonable. For further details on the composition algorithm used by ASG, please refer to Meyer and Weske (2006). In case of successful composition or a perfect matching of an individual service, an abstract service composition representation is created and forwarded to the next subcycle. Otherwise, the processing of the semantic service request is aborted and an error message is sent to the requester.

Abstract service compositions are transformed into enactable service compositions during the Binding Subcycle. This happens by binding the semantic service specifications – which act as placeholders – to concrete services. There are two different strategies for service binding implemented in ASG: binding via selection and binding via negotiation. The first strategy – binding via selection – is quite simple, but it works without special extensions to the usual Web services: For each semantically described placeholder, the platform discovers services which are matching the functionality defined for the placeholder. If there are several services matching one placeholder, then the platform selects the service that is best fitting to the optimization criteria, which can be specified as a part of the semantic service request. The second strategy works only with extended services which are supporting a negotiation interface. For each placeholder, the platform starts a negotiation on negotiable service properties with all matching services. The platform tries to find a combination of services which fits as good as possible to the desired properties of the semantic service request. It is worth mentioning that the properties of services which are discussed now are not the semantic functionality of the service like the above mentioned input, preconditions, output and effects of a service. Rather the properties describe qualities of the service execution, like, for example, the duration or the costs per execution. These properties are limited insofar that they are not allowed to depend on the concrete input data. When an agreement with a particular service is achieved, a digital contract is set up and signed by both parties (Momotko et al., 2006).

The third step in processing of semantic service requests is the Enactment Subcycle. It receives enactable service compositions from the Binding Subcycle and enacts them by successively or – where possible – invoking the scheduled services in parallel. The invocation of services is monitored by the platform for two reasons: on one hand, it is monitored to verify if the previously contracted peculiarities of properties are met (Momotko et al., 2006). On the other hand, the monitoring data is passed to a profiler which aggregates the data to a service-specific profile. This profile contains an aggregate of experiences made during past invocations of a service. It contains informations like the average service execution time or reliability in the sense of observed probability of failure. This profile information can be used by the Planning and Agreement Subcycle to avoid unreliable services. After successful enactment of the service composition, the result is collected and is sent back to requester.

The ASG service delivery life cycle includes two mechanisms to handle dynamics when it comes to explicitly considered or unconsidered failures of services: rebinding and replanning. The definition of the term failure used here is compliant to the definition in Avzioni 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 (destruction) of a package by a package shipping service as it might happen due to a car accident or theft. Considered failures are handled in ASG by conducting replanning. In case, a considered failure occurs during the invocation of a service, the Planning Subcycle is triggered to find a new composition. Referring the package shipping service, this might mean that a new package is seized and sent. Naturally, not all considered failures can be handled by recomposition. In case of the credit card withdrawal service, a recovery is not possible and it is even not useful if a credit card is turned out as being invalid. An alternative to the use of replanning to handle considered failures would be to include all possible, considerable failures in the initial service composition. There are two reasons why we did not choose this alternative in ASG: Firstly, the planning of all possible, considerable failures takes significantly more time and resources than an optimistic planning. Furthermore, the resulting composition is greatly expanded by many additional paths which are intended to handle the considered failures, but which have a low probability of being used. Secondly, the planning of all possible, considerable failures also mean that a failure in the recovery of failures is also to be taken into account. In some cases, this may lead to situations where a service composition grows up to an infinite size. Such situations can be modelled in a finite way by using boundless loops, but our service composition algorithm is not capable of detecting loops and applying them on a service composition.

In contrast to the considered, are the unconsidered failures not explicitly specified for a service. 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 happens at a given point of time. For this reason, it assumes in such cases of failure, that the corresponding service simply has not been executed and thus its desired results and effects are not achieved. Such unconsidered failures are handled in two phases. In the first phase, the platform tries to recover the failure by rebinding. This triggers a new pass of the Binding Subcycle. A search

2
for an alternative equivalent service to the already invoked and failed service is conducted by negotiation 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 replanning the composition in the Planning Subcycle. In case this planning is successful, the Binding Subcycle is invoked. After the outstanding service placeholders have been bound to concrete services, the new composition is enacted in the Enactment Subcycle.

6 Application scenario

The scenario we present here has been successfully implemented by the ASG project on the basis of existing ‘real-world’ services and the ASG platform reference implementation. We have prepared a demonstrator on the ASG project homepage which implements this scenario. The demonstrator can be downloaded for free, and provides an easy-to-use installer for Windows platforms to aid the testing of the platform.

The example is based on the 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 ISP uses the ASG platform in a role as a Hosting Provider to bundle and integrate external basic services provided by, for example, registrars like Denic, VeriSign and Directi and by digital payment providers like SaferPay or PayPal (refer to Figure 4). The integrated services are provided by the ISP to Resellers who use the services either internally or implement a proper user interface for the end user or consumer. 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 life cycle as shown in Figure 3.

Another benefit the reseller has from using the ASG platform is the automated failure handling. As already described the ASG platform supports the handling of failures by rebinding and replanning. Figure 5 shows how a service composition is recovered in case a service like Denic CheckDomain fails, for example, because of network problem. In this case, the platform searches for a semantically equivalent service (VeriSign CheckDomain) and replaces the failed service. Figure 6 shows how a failed payment service from PayPal is replanned and replaced by two services from SaferPay. Both, rebinding and replanning have been implemented and tested on the ASG reference platform in the ASG project.

Figure 5 Service re-binding example

Another benefit the reseller has from using the ASG platform is the automated failure handling. As already described the ASG platform supports the handling of failures by rebinding and replanning. Figure 5 shows how a service composition is recovered in case a service like Denic CheckDomain fails, for example, because of network problem. In this case, the platform searches for a semantically equivalent service (VeriSign CheckDomain) and replaces the failed service. Figure 6 shows how a failed payment service from PayPal is replanned and replaced by two services from SaferPay. Both, rebinding and replanning have been implemented and tested on the ASG reference platform in the ASG project.

Figure 6 Service re-planning example

7 Related work

The Web Service Execution Environment (WSMX) is a software system 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. Support for manual composition of services is currently under consideration (Haller et al., 2005).

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 runtime 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 rebinding of individual services at runtime to master service failures. However, an automated composition of services and negotiation of non-functional properties is not supported (Verma et al., 2005).

8 Conclusion

This contribution argues that the full potential of service-based software architectures can only be achieved by extending the currently available syntactic descriptions of services with semantic descriptions. Such an extension facilitates the dynamic matchmaking and composition of services by means of service descriptions to realise added value services and to make them available to the market through applications. Prototypic applications like the ASG platform are trending projects in the way of completely exploiting the potential of service based software architectures. To mobilise this long term goal in practice, a close cooperation between research institutes and the software industry is required. To foster future reuse of the ASG platform, major parts of the platform are published under an open-source licence at the project homepage.

Acknowledgement

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

References


Notes

1ASG is supported by the Sixth Framework Programme of the European Commission, contract number 004617, call identifier FP6-2003-IST-2.

2For example, If package shipping fails with a package loss, then create a new package and ship it again. However, also, this package shipping may fail with a package loss, therefore again, a new package has to be created and shipped again. However, also, this package shipping may fail with a package loss, therefore...

3http://asg-platform.org/cgi-bin/twiki/view/Public/PrototypeDemo.