Semantic Enterprise Services Platform: Motivation, Potential, Functionality and Application Scenarios

Dominik Kuropka, Anja Bog, and Mathias Weske
Hasso-Plattner-Institute at the University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
{dominik.kuropka | anja.bog | mathias.weske}@hpi.uni-potsdam.de

Abstract

Service oriented software architectures will form the core of operational enterprise IT landscapes in the future. This contribution starts with an introduction of the state of the art in service oriented architectures. A concrete case study identifies central requirements that are not satisfied by these architectures so far. 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 possible application scenarios are outlined.

1 Introduction

Today's software architectures in business environments are challenged to meet the changing requirements of the market rapidly and cost-efficiently. Because of realizing considerable competitive advantages by reacting swiftly and adequately to new requirements, the development of appropriate systems and technologies plays a decisive role in the design of software architectures in enterprise environments. A central property 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 present a promising approach for this. It is assumed that they will form the core of operational IT landscapes in the future.

This contribution introduces the state of the art in the domain of service oriented architectures. A concrete case study identifies central requirements that are not satisfied by these 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[1]. 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. Additionally possible application scenarios will be outlined.

2 State of the Art

Increasing pressure of competition in the software industry forces system providers to expand their clientele continuously to achieve cost reduction by measures of scale in software development and maintenance. To acquire more customers, enhancing offered functionality and making existing functionality accessible more easily is of importance. These requirements in general result in increased complexity and hence in a disproportionate increase of costs for software development and maintenance, in case this bias is not met with strategies of complexity reduction. A technical and in many scenarios successful strategy is the separation of functionality into appropriate, reusable and if applicable distributed components that realize the systems functionality. Known technologies for the implementation of such component based systems are Remote Procedure Calls (RPC) [1], the Common Object Request Broker Architecture (CORBA) [2] and Web services [17, 18].

For a long time, the separation of software systems into

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

reusable components has been done by software developers almost exclusively along technical aspects. This is particularly appropriate in case the components belong to a single organization. However the demand for short response time to market demands (e.g. flexible adaption of whole supply and production chains to short-dated fluctuations of demand) increasingly forces the integration of external information systems with internal information technology. 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.

A service (in this context) is a well defined functionality that is provided on electronic request. To apply services a Service Oriented Architecture (SOA) is required. It defines the following roles of the participants [4]: The *service provider* is the institution that offers a service by registering it with a service broker. It is responsible for executing the published services and thus has to provide an adequate infrastructure. A *service broker* offers a directory for registering new services and retrieval of existing services. It might be the same institution as the service provider or requester. The *service requester* is the institution that asks for a service. To find an applicable service the service requester asks the service broker. If such a service is found, the services specification contains information about the service provider and technical details about how to invoke the service.

Usage of services has two advantages for providers as well as requesters: reduction of complexity and improved integrability. The reduction of complexity results from the fact that service oriented architectures separate the whole functionality of the system into several encapsulated part functionalities and make them individually available. The adaption of systems towards business environments takes advantage of the fact that services are constructed, encapsulated and documented along business aspects and not mainly along technical aspects.

Regarding technical issues services may be implemented using different technologies that allow invocation of component functionality over networks. Comparatively widespread are standardized Web services. As base of communication they mostly use the established Hypertext Transfer Protocol (HTTP) [5] known from the area of the World Wide Web. As a universal data format the Extensible Markup Language (XML) [6] is used that features high flexibility and broad availability of tools and libraries for processing XML documents. Service calls are generally handled by using the Simple Object Access Protocol (SOAP) [3]. In addition to invocation and communication protocols standardized formats for description and matchmaking of Web services are defined. Web services are usually described by a document utilizing the Web Service Description Language (WSDL) [7]. This document describes how service calls are technically realized. It contains information about the server the request needs to be sent to, the name of the method that is going to be invoked as well as the XML schema the input and output data (messages) have to correspond to. Such WSDL documents are stored at the service broker at the registration time of the service. A standard for the communication with service directories is also specified: Universal Description, Discovery and Integration (UDDI) [8]. This standard allows services to be specified and searched for by means of business categories and service types (tModels). Additionally service descriptions using natural languages can be stored. 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 of the next service directly without conversion or change. Since computers are not able to "understand" the processed data in terms of human thinking, 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 professional employees, who understand the technical aspects as well as the business dimensions of the data and the underlying processes.

A concrete, however simple integration example for such a problem is described 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 realized by 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 required service 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 in general 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 endeavor. 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 suitable 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.

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 is described. A formal description of the functionality of a service and the meaning of its data structures is 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 labeled 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 [9, 10]. 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 [11]. Furthermore, a model is a representation of a system of objects for the purpose of a subject. It is constructed by humans (modelers) and is represented by a system of symbols (modeling language). 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 Modeling Language(WSML) [13] in Europe and the Web Ontology Language (OWL) [14] 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 *Adaptive Services Grid (ASG)*[2] project. Our platform discovers and composes services to subprocesses using semantically specified data structures and services, based on domain ontologies using WSML. These specifications serve as a starting point for the specification of the

[2] `http://asg-platform.org`

semantics of data schemes that are the basis of potentially different services. 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

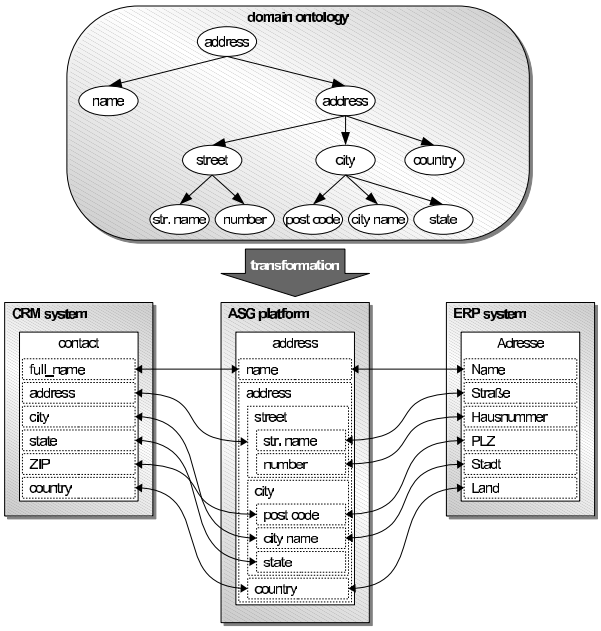

Figure 1. Mapping of Ontologies and Data Structures

demonstrates a simplified visualization of a section of a domain ontology referring to the above mentioned integration example and how the data structures of the ontology are represented as XML schema. Furthermore it shows how the data structures of the above mentioned sample CRM and ERP systems 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 to be implemented depending on the number of systems or interfaces to be integrated grows in a fast disproportionate manner. See Figure 2 for comparison.

A service needs to be registered to become applicable within the platform. At registration time the following information about a service will be deposited: the semantic service specification, non-functional properties of the service and 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.

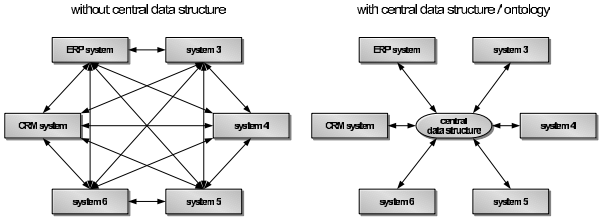

Figure 2. Mapping of Data Structures without/with a Central Data Structure

This allows to specify a service to have a telephone number as input and an address as output data 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, also having 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 be the means of natural language documentation, which is not processable automatically.

Figure 3 depicts a simplified visualization of the semantic specifications of the services from the integration example (service 1 and 2). Service 1 is specified as a service receiving a telephone number as input. As output it returns an address related to the received telephone number. Additionally the service assures that only customer addresses are returned. Service 2 on the other hand needs the address of a customer (customer address) as input and returns an order, whereas this order is directly displayed on the customer advisers monitor. Service 3 is of the same syntactical type as service 1 regarding input and output data, but instead of returning a customers 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 presented and distinguished using formal semantic definitions.

5 Life Cycle of Service Delivery

The processing of a call and the delivery of a service by the platform differs from the approach of current service platforms. In present systems a certain service invocation is always relayed to a previously statically bound service. The semantic service provisioning platform implemented in the ASG-project forwards the invocation to a composition

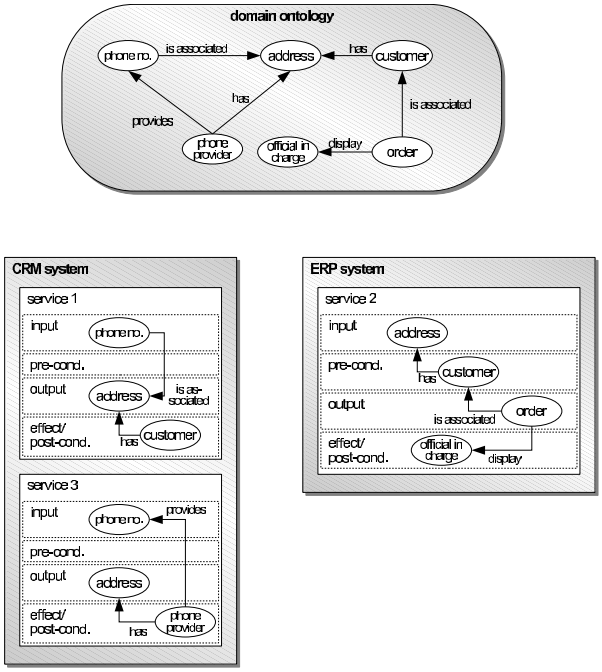

Figure 3. Semantic Specification of Services

component. This component automatically composes a new service from several services offering parts of the needed functionality. To raise the performance of the automated composition we implemented a heuristics as described in [19].

5.1 Planning Sub-Cycle

The service request sent to the platform consists of multiple semantic building blocks: starting state, goal state and probably side conditions for non-functional properties and optimization criteria. Building blocks that are of highest importance are the starting state and goal state, because they define the semantic functionality of the desired service. The starting state comprises the semantic description of the actual situation which is the given base for a service. This includes in particular the already given data relevant for the desired service that have to be part of the concepts of the domain ontology.

The goal state describes the desired result of the service call that has to be reached from the given starting state. It usually contains a list of the desired data structures or concepts of the ontology and effects. Accordingly the starting state and the goal state are analogical to the semantic specification of a service. However they do not represent the functionality of an existing, but rather the functionality of a desired service. In addition to the wanted functionality restrictions on non-functional properties may be defined. These restrictions might be used for example to prevent or to assure services of certain providers to be chosen for execution by the platform, because of security issues. Alternatively any other restrictions like processing time or costs may be defined on properties of services that are modeled in the domain. Optimization criteria (for example minimum processing time or least costs) serve to acquaint the platform which services or service compositions to choose in case several functionally equal and matching services or service compositions are available.

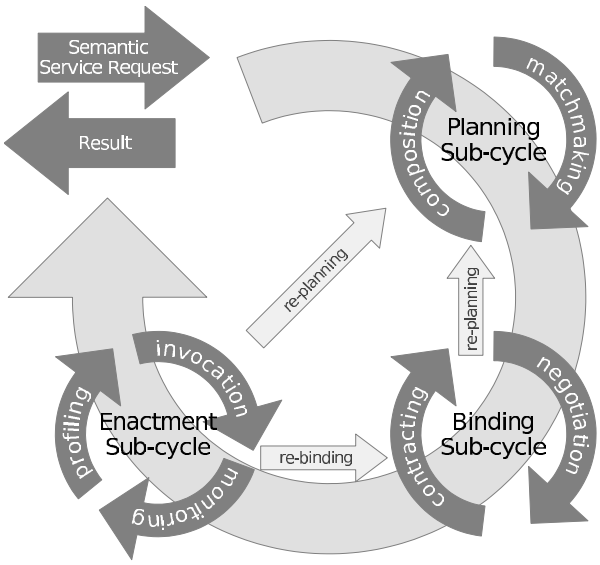

Figure 4. Service Delivery Life Cycle.

Incoming service requests are processed by the platform in three steps as can be seen in Figure 4. The first step is the planning sub-cycle. In the simplest case it searches for a service that completely accomplishes the request. If such a service is not available the system tries to find a service composition that is beginning with the starting state leads to the goal state. Composing services is also done stepwise by the platform firstly identifying services that are executable in the starting state because of their preconditions being fulfilled. For each one of these services the expected state after its execution is computed by the platform. This operation is called virtual execution, because the services are not actually executed, but the state after the execution is estimated using their semantic specification. Considering this computed state the platform evaluates if it is closer to the specified goal state. The best service regarding this evaluation is chosen by the platform and added to a preliminary service composition. Based on the current estimated end state new services are searched for that are executable and whose end state is closer to the desired goal state. These steps are carried out until reaching the goal state. A full description of the heuristics used is presented in [19].

The service composition is stored as an extended

BPEL4WS document[16] in the platform. Extending this document format allows for storing semantic descriptions for the services instead of firmly bound services. Thus reusability and flexibility of service compositions is enhanced, since deciding which concrete service is going to be bound is done during run time of the service composition. Consequently the decision of binding a concrete service is kept dynamic and flexible and simultaneously a composed or manually modeled service composition can be reused without having to go through the whole planning sub-cycle again. For frequently recurring requests this is especially advantageous.

5.2 Binding Sub-Cycle

In the next step called binding sub-cycle the appropriate services according to the semantic descriptions of the services contained in the service composition from the first step are searched for and bound to the composition. The binding sub-cycle also regards dynamic non-functional properties in contrary to the planning sub-cycle that merely considers static non-functional properties. During the binding sub-cycle all functionally appropriate services for each service specification contained in the service composition are sought out. Afterwards dynamic non-functional properties are queried or negotiated if supported by the service. On the basis of this information a combination of services is chosen during the binding sub-cycle that meets possibly defined restrictions concerning non-functional properties on the one hand and on the other hand fulfills the optimization criteria preferably well. Finally the chosen services are bound to the service composition that is now stored in the system as a normal BPEL4WS document. In the case that functional properties of a single service have been negotiated, a service level agreement is drawn up by both sides and kept in the platform. As in the planning sub-cycle bound service compositions of frequent or time-critical requests may be cached or manually provided if necessary.

5.3 Enactment Sub-Cycle and Exception Handling

In the last step called enactment sub-cycle the bound service composition is executed. Single services of the composition are either executed in sequence or if possible in parallel. Invocation of the services is done using the service proxy that transforms the input data to the required XML data schema if needed and also transforms the results to the data schema based on the domain ontology. After execution of all services contained in the composition is completed the goal of the request is usually reached and the result is returned to the service consumer who has invoked the service.

Since the execution of a service may fail due to many reasons the platform implements a three-stage handling of errors and exceptions (exception handling). The first stage is embedded in the enactment sub-cycle. With its help smaller communication errors of the types “connection aborted” or “server not available” can be corrected, as far as they can be resolved by re-initiating the communication connection or using an alternative server of the affected provider. Such a correction neither affects non-functional properties nor the structure of the semantic service composition of the invocation en bloc. Cases that can not be handled by the first stage, for example if a connection to a server of a specific provider cannot be established because of a lasting network failure, are forwarded to the second stage of error handling.

The second stage is implemented in the binding sub-cycle and a new alternative provider for a concrete semantically specified service block of the service composition is sought. If a provider is found, it will be bound to the service composition at this position and the execution is returned to the enactment sub-cycle for further processing. Since a new provider has been chosen the non-functional properties of the whole service composition have been changed, whereas the basic structure of the composition is maintained. Besides the non-functional property “provider” other non-functional properties like “price” or “processing time” of the service execution may have changed, too. However these changes may only go as far as the restrictions on the non-functional properties as defined in the original request are not violated. Consequently this stage may also not be able to handle the error if either an alternative service provider cannot be found to substitute a faulty service or the conditions of a possible substitution violate the defined restrictions. In this case the error is passed to the planning sub-cycle for re-planning of the whole service composition.

The third stage tries to use the currently achieved state of the service composition as a given starting state and attempts to reach the defined goal state from this state without using the corrupt service. This approach is based on the assumption that the faulty service can be substituted by a composition of multiple smaller services or can be completely eliminated by using a different procedure. If such a composition consisting of semantic service specifications is found, it will be passed to the binding sub-cycle as usual. Using this approach changes the non-functional properties of the entire composition as well as the composition itself is modified. Whereas the adherence to the restrictions of the original request still needs to be assured. In case the third stages attempt to fix the error also fails, the platform is not able to handle the error automatically and an error message is sent to the service consumer, who has invoked the service.

The three stage error handling as described above deals with technical errors and exception states that are not part

of the semantic service specifications. Besides technical exception states mostly based on communication errors services may have semantically relevant exception states. An example for a semantically relevant exception state for the service "debit credit card" may be the credit card being void or expired. Another example concerning the service "send package" would be the exception state of the package being lost. If an automatic handling of these states is wanted, they have to be modeled as possible effects of a service in the semantic service specification even if this behavior is not desired. In current process and workflow management systems exception states of a process or workflow are ideally modeled in advance. Accordingly processes usually become huge, complicated and hard to maintain on the long run. Another solution for this problem is possible using the approach of automated service composition and appropriate re-planning. The platform assumes an optimistic scenario when planning service compositions, which means that initially only wanted effects will be planned. Hence compositions are smaller in contrary to a complete plan of all possible exception conditions, which increases the efficiency of the composition procedure. In case a service does not yield the wanted effect during execution in the enactment subcycle, but results in a semantically specified exception state, the partly executed service composition is directly returned to the planning sub-cycle for re-planning. Based on the state in which part of the composition was executed successfully and the exception state of the problematic service the platform attempts to compensate the exception state and reach the original goal by re-planning of the remaining steps and addition of new steps. Failure is still possible for example caused by the absence of appropriate services for a compensation of the exception state or caused by restrictions of their invocation. In this case a manual solution of the problem has to be taken into account if it existent.

Besides the execution of services and service compositions the enactment sub-cycle offers tools for automatic monitoring of services. Thus reliability and reachability of services as well as successful delivering of results can be supervised. A profiler may eventually use the monitoring data to evaluate a concrete service or service provider. The results of this evaluation can be used for service composition in the planning sub-cycle to rule out non-reliable services and providers in the future.

6 Application Scenarios

A service platform offering semantically annotated services might be applicable as a market place for services as well as in the integration of intra-company and external services. In the case of the integration scenario depicted in Figure 5 the semantic services platform takes on the role of an intelligent mediator and integrator operating on a high abstraction level. Services of different integrated application systems like ERP or CRM can be invoked and combined to an integrated application system using the platform. Additional functionality offered by the platform is the possibility to integrate external services transparently into the system and hence make them available to all internal systems. In this context the adaptive properties of the platform are of special use. By automated planning and binding of services the optimal services (for example regarding costs) can be used at any given time. Furthermore the three-stage error handling strategy of the platform allows for compensation of failures of single services during run time if they can be substituted. Owing to the platform the usage of external services is simplified, because after registration they are used by the platform automatically if they match the existing semantic context of the applications and if they are competitive regarding their non-functional properties. The registration and semantic adaption of services as well as providing an appropriate ontology is the task of the company using the platform as integration tool. Hence the company has full control over the used services, which is of advantage regarding security aspects and form of contract. The operator of the platform can choose service providers according to his requirements and can (but does not have to) negotiate and enforce contracts with them.

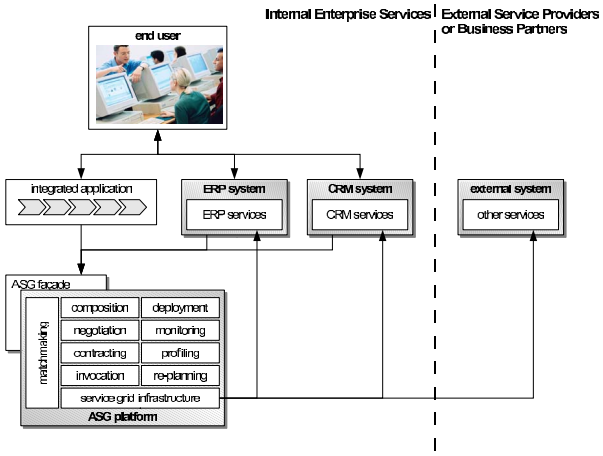

Figure 5. Integration Scenario for the Semantic Services Platform

Apart from the medium term field of application as integration platform, the platform offers to be a market place for services in the long run (see Figure 6). This scenario includes four groups: The basic service providers offer fundamental services that are registered with the market place, which is operated by the market place owner. The market place owner provides the platform as a market place, guarantees trust and is responsible for maintenance and provision of an ontology. An important task is the verification

if registered services actually offered meet their described functionality. In this context the monitoring and profiling functionality of the platform are useful. The platform on the other hand provides its registered services to the end service providers. The end service providers use these services to build light weight applications that are then offered to institutional or private end service consumers. Two approaches are faced regarding an implementation of the form of contract between the parties. In the first approach the basic service providers and the market place owner conclude a contract concerning the modalities of the offered services and the market place owner also concludes a contract with the end service providers concerning usage of the services. In this solution the market place owner bears the liability risks in case of a faulty service execution, which however entails the advantage of increased trust in the market place. Alternatively the market place can merely be an intermediary for the offered services. In this case a digital form of contract is required in order to establish contracts between the end service providers and the basic service providers regarding the usage of services during run time. This second alternative, being more interesting in the long term with respect to facilitating a very dynamic and flexible platform for electronic services, places high requirements on the market place owners infrastructures for security, encryption and signatures. Finally it should be noted that the above presented market place scenario is a long term vision which needs more detailed elaboration in the future to analyze its practicability.

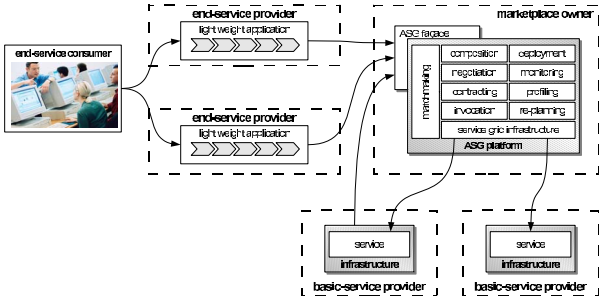

Figure 6. Market Place Scenario

7 Conclusion

This contribution argues that the full potential of service based software architectures can only be reached 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 realize added value services and to make them available to the market through applications. Prototypic applications like the Adaptive Services Grid are trend setting projects on the way of completely exploiting the potential of service based software architectures. To mobilize this long term goal in practical a close cooperation between research institutes and the software industry is required from the current point of view to combine technical and academic expertise in the context of service based software architectures.

Acknowledgment: This article contains results that have been acquired in the context of the Adaptive Services Grid project (contract number: 004617, identifier: FP6-2003-IST-2).

References

[1] Andrew Tanenbaum: Modern Operating Systems. 2nd Edition, Prentice Hall, 2001.

[2] Object Management Group: Common Object Request Broker Architecture: Core Specification. Version 3.03. 2004. http://www.omg.org/docs/formal/04-03-01.pdf

[3] World Wide Web Consortium: SOAP Version 1.2 Part 0: Primer. 2003. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/

[4] Steve Burbeck: The Tao of e-business services. IBM Corporation, 2000. http://www.ibm.com/software/developer/library/ws-tao/index.html

[5] World Wide Web Consortium: Hypertext Transfer Protocol – HTTP/1.1. 1999. http://w3c.org/Protocols/rfc2616/rfc2616.html

[6] World Wide Web Consortium: Extensible Markup Language (XML) 1.1. 2004. http://w3c.org/TR/2004/REC-xml11-20040204/

[7] World Wide Web Consortium. Web Services Description Language (WSDL) 1.1, 2001.

[8] Organization for the Advancement of Structured Information Standards (OASIS). UDDI Version 2 Specifications, 2002.

[9] Smith, B.: Ontology and Information Systems, 2002. http://wings.buffalo.edu/philosophy/faculty/smith/articles/ontologies.htm

[10] Ziga, G.: Ontology: Its Transformation From Philosophy to Information Systems. In Proceedings of Formal Ontology in Information Systems. 2001, pp. 187197

[11] Kuropka, D.: Modelle zur Reprsentation natrlichsprachlicher Dokumente Information-Filtering und - Retrieval mit relationalen Datenbanken. In der Serie: Advances in Information Systems and Management Science, 10. Ausgabe. Logos Verlag, Berlin, 2004.

[12] Becker, J.: Integrationsorientierte Wirtschaftsinformatik. http://www.wi.unimuenster.de/is/aws60.de/becker/modell.htm

[13] de Bruijn, Jos (Editor): The Web Service Modeling Language WSML, 2005. http://www.wsmo.org/TR/d16/d16.1/v0.21/

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

[15] World Wide Web Consortium: XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/xslt

[16] Andrews, T; Curbera, F.; Dholakia, H; Goland, Y.; Klein, J; et. al.: Business process execution language for web services version 1.1. Technical report, BEA Systems, IBM, Microsoft, SAP AG and Siebel Systems, 2003.

[17] World Wide Web Consortium: Web Services Activity. http://www.w3.org/2002/ws/

[18] Gesellschaft fr Informatik: Informatiklexikon: Web-Services. http://www.gi-ev.de/service/informatiklexikon/informatiklexikon-detailansicht/meldung/95/

[19] Meyer, H.; Weske, M.: Automated Service Composition using Heuristic Search. In: Proceedings of the Fourth International Conference on Business Process Management (BPM 2006), Vienna, Austria (2006).