Hasso Plattner Institute
for Software Systems Engineering

Final year bachelor project
WS 2005/2006

Semantic SOA – Realization of the ***Adaptive Services Grid***

February 28, 2006

Bastian Steinert, Jan Möller, Philipp Sommer,
Sebastian Steinhauer, Stefan Hüttenrauch,
Tobias Queck, Torsten Hahmann

Table of Contents

1 ACKNOWLEDGEMENT

Here we present the result of the final year bachelor project "Semantic SOA – Realization of the Adaptive Services Grid". The work on this project lasted from September, 1, 2005 to February, 28, 2006 and was conducted at the Hasso-Plattner-Institute for Software Systems Engineering at the University of Potsdam under scientific supervision of the chairs for "Business Process Technology" (head: Prof. Mathias Weske) and for "Operation Systems and Middleware" (head: Prof. Andreas Polze). For their contributed scientific expertise we especially thank Dr. Dominik Kuropka, Guido Laures, Harald Meyer, Peter Tröger and Dr. Martin von Löwis. The project is part of the integrated research project "Adaptive Services Grid" (ASG) sponsored under the sixth framework program of the European Commission[1]. We want to thank all ASG partners for their cooperation and valuable feedback that helped to succeed with the project. We especially appreciate the support and advise of our industrial partners DaimlerChrysler Research, Ulm and NIWA Web Solutions, Vienna. With DaimlerChrysler Research, Dr. Ingo Melzer and his research team as well as Dr. Ralf Hinz and his team contributed their valuable know-how and supported us in our project's work. At NIWA Web Solutions, we were helpfully advised by the whole team and in particular by Bernhard Peissl and Alexander Wahler.

1 http://asg-platform.org, EC Contract No. 004617

PART I: PROJECT OVERVIEW

2 ROADMAP

Because of the complexity of ASG and the great amount of ongoing research our bachelor project addresses different topics. These topics are partly independent from each other. However, they serve the common goal to get a running ASG demonstration platform using an industrial scenario. With this project we

(1) show the flexibility of the ASG reference architecture in respect to exchangeable individual components and at the same

(2) we prove the usability for real world applications.

We aim to demonstrate ASG's flexibility using the IBM products Websphere Application Server 6 and the database management system DB2 V8.1 within the service grid infrastructure. Additionally we replace the current process execution engine provided by Rodan Systems. Instead, we integrate the open-source workflow engine PXE with enactment component.

In cooperation with our industrial partner NIWA Web Solutions, Vienna we

(3) identify, define, and implement a scenario in the field of dynamic supply chains for internet service providers.

For that purpose we must identify appropriate service providers and specify service functionality as well as semantics in the ASG context. External functionality must be wrapped by Atomic Services in order to dock them into the ASG platform.

The previous ASG demonstration prototype does not yet suffice our needs for dynamic service composition, selection, and execution. The goal of a fully functional demonstrator requires us to extend the main components discovery, composition, negotiation, and enactment and to integrate them appropriately.

3 Documentation Structure

During the bachelor project we documented our efforts and achievements in the following artifacts:

- Dynamic Supply Chain Scenario for Internet Service Providers.
- Open-source Workflow Engine Integration into ASG Platform.
- Integration of a Mobile Service Provider into the ASG infrastructure.
  Work at DaimlerChrysler Research, Ulm (REI/VA) August and September 2005.
- Use Case "Automated Garage Service" for Integration of a Mobile Service Provider into the Adaptive Services Grid. *Textual Version.*
  Work at DaimlerChrysler Research, Ulm (REI/VA) August and September 2005.
- Use Case "Automated Garage Service" for Integration of a Mobile Service Provider into the Adaptive Services Grid. *Modelled Version.*
  Work at DaimlerChrysler Research, Ulm (REI/VA) August and September 2005.
- Migration of Service Grid Infrastructure from JBoss/Postgres environment to an IBM environment (Websphere/DB2).
  Work at DaimlerChrysler Research, Ulm (REI/ID) August and September 2005.
- User Guide MexMan4AS.
- Paper: "An adaptive solution for internet services' supply chains".
  1st Industry Workshop on Semantic Systems at Semantics 2005, Nov. 23.-25. 2005.

4 CHRONOLOGY

Period	*Main actors*	*Tasks*
Aug. - Sept. 2005	Bastian Steinert, Sebastian Steinhauer	Work at DaimlerChrysler Research, Ulm: Migration of Service Grid Infrastructure from JBoss/Postgres environment to an IBM environment (Websphere/DB2)
Aug. - Sept. 2005	Stefan Hüttenrauch, Tobias Queck	Work at DaimlerChrysler Research, Ulm: Integration of a Mobile Service Provider into the ASG infrastructure and definition of use case "Automated Garage Service"
Sept. 2005	Jan Möller, Philipp Sommer, Torsten Hahmann	Open-Source Workflow engine evaluation for use with enactment
Okt. - Nov. 2005	Jan Möller, Philipp Sommer, Torsten Hahmann	Work at NIWA Web Solutions, Vienna: Scenario definition and development of service landscape
Okt. - Nov. 2005	Bastian Steinert, Sebastian Steinhauer, Stefan Hüttenrauch, Tobias Queck	Main development phase of Eclipse plug-in MexMan4AS Setup of autonomous testbed infrastructure with Websphere Application Server 6, JBoss Application Server 4, database management systems DB2 V8.1 and PostgreSQL 8.1
Nov. - Dec. 2005	Jan Möller, Philipp Sommer, Torsten Hahmann	Scenario description and presentation of scenario development methodology in paper for Semantics 2005, Vienna.
Dec. 2005	all	Setup and demonstration of prototype "C4/C5 Integration" (enactment and service grid infrastructure)
Dec. - Jan. 2005/2006	Sebastian Steinhauer, Stefan Hüttenrauch, Tobias Queck	Atomic Service development MexMan4AS plug-in extension
Dec. - Jan. 2005/2006	Torsten Hahmann	Semantic Service Descriptions and Ontology development using Flora2
Jan. 2006	Bastian Steinert, Jan Möller, Philipp Sommer, Torsten Hahmann	Implementation of negotiation
Jan. - Feb. 2006	all	Integration and testing of ASG main components Incremental development of test cases for integration of all developed Atomic Services

4.1 Tool Support for Atomic Service Development – MexMan4AS

Because for Atomic Service development a lot of knowledge about several technologies, especially in the realm of J2EE and build management is necessary, we decided to support Atomic Service developers and ourselves by implementing an Eclipse plug-in. This plug-in manages the most important tools (Maven and Xdoclet) and builds skeleton code used for Atomic Service development.

Part of the documentation is a user guide (see document *User Guide MexMan4AS*) for this plug-in facing miscellaneous aspects of Atomic Service development and explaining how to work with the latest version of MexMan4AS. The plug-in itself can be downloaded from http://tb0.asg-platform.org/~steinhauer/.

PART II: Evaluation and Implementation of ASG Architecture Concepts

5 Discovery

The ASG discovery component is responsible for executing queries on the semantic service descriptions of the Atomic Services. These semantic descriptions are based upon an ontology that describes the business concepts of a limited domain and relations amongst those concepts. The discovery component encapsulates a Flora instance to which requests in logic expressions (F-Logic/Flora) are forwarded to extract knowledge about abstract services functionalities and concrete invocation protocol of particular services.

When using a real discovery database, it is required to load all service specifications, the underlying ontology (which may consist of several partial ontologies), and the basic ASG ontology for service discovery. During the bachelor project efforts the discovery component has been changed in a way that, upon loading the service specifications, all included ontologies (through the flAdd command in Flora) and the transitively included ASG ontology will be loaded as well. The now fully functional discovery database allows us to replace the previous mock links of other ASG components to discovery and integrate negotiation and composition components with the discovery component.

The overall architecture of the discovery component containing the Flora reasoner and the methods for extracting semantic service specifications, service properties, and groundings has been changed to enforce the use of a single Flora instance. The reasoner as singleton avoids problems that might occur when running more than one Flora instance concurrently. On the other hand, the reasoner must additionally ensure that all method calls run isolated from each other. This is satisfied by using Java synchronization mechanism in critical sections.

5.1 Restrictions

Discovery is dependent upon the Flora reasoner. The reasoner component is realized using an java-prolog interface (called Interprolog) that interacts with the external Prolog interpreter XSB, which has been extended to process Flora queries. The set-up of XSB with Flora and an appropriate Interprolog interface is dependent upon the operating system. We have been able to set-up a reasoner on Windows as well as Linux. However, even on these systems reasoning is quite far from being stable. Since the Interprolog interface is not open-source, debugging is difficult. Uncaught exceptions in the Java VM leads to abnormal exits (no stacktrace will be displayed, exit with memory access violation).

Performance is another open issue in the discovery component. Requests on the Flora reasoner take up to several seconds depending on the complexity of queries. More complex queries must be used with great care and only if unavoidable. The queries implemented by the discovery component still need optimization. Most of the time for processing an ASG request is consumed by the discovery component. Currently a composition of five services takes about one minute starting from negotiation. Since composition heavily interacts with the discovery component, more than five minutes must be estimated even for a simple composition of three services.

6 NEGOTIATION

Within the ASG execution sequence the negotiation component accomplishes its tasks after composition and before enactment. The input for this component is an execution plan based upon semantic invocations, semantic variables, and assignments. In particular, such a plan does not reference concrete service implementations. Instead the semantic invocations are specified with a set of precondition and effects.

Negotiation is responsible for selecting concrete service implementations according to the condition set of each semantic invocation. Afterwards negotiation converts all semantic variables and assignments according to the groundings and interface definitions of the selected service instances.

In short, following tasks must be fulfilled by the negotiation component:

- Retrieval of service groundings for each semantic service invocation from discovery component
- Instantiation of all selected services to a logical instance by using the service factory
- Selection of an appropriate service grounding for each semantic invocation
- Attaching selected services' WSDL, which have been retrieved by service instantiation, to the execution plan
- Replacement of semantic variables and assignments with service variables derived from specified grounding (prerequisite: consistent naming for ontology concepts referenced in service specifications and WSDL message parameters of concrete services)

Both negotiation and enactment need WSDL data, that is now part of the execution plan (composed service). The negotiation component needs the interface definition of the web services for generation of a BPEL compliant execution plan. The enactment component needs the WSDL information for service invocation. Especially, the execution engine requires binding information (e.g. the URL of the endpoint port) expressed in the WSDL data. Previously WSDL could not been retrieved from the invocation layer, it had been stored separately in higher ASG layers. Now, the C5 service factory directly returns the whole endpoint reference document during service instantiation. The endpoint reference document will be attached to each partner link according to the standardization draft Web Service Addressing 1.0 – WSDL Binding[1]. It consists of an identifier for the created logical service instance, an endpoint address where the request must be sent to, and the mentioned WSDL document.

6.1 RESTRICTIONS

Currently, negotiations returns the first service that fulfils the given condition (precondition and effects) and can be instantiated successfully. Those instances must be used for negotiation in future. The selection of an appropriate service grounding for each semantic service invocation must be based upon negotiation of Quality of Service properties.

7 ENACTMENT

The Enactment component demonstrates the flexibility of the ASG reference architecture. We replaced the previously used enactment component with a component based on the

1 http://www.w3.org/TR/ws-addr-wsdl/

open-source workflow engine PXE. The evaluation procedure and all required integration tasks are documented in the artifact "Open-Source Workflow Engine Integration into ASG Platform".

8 Testbed Infrastructure and Units of Deployment

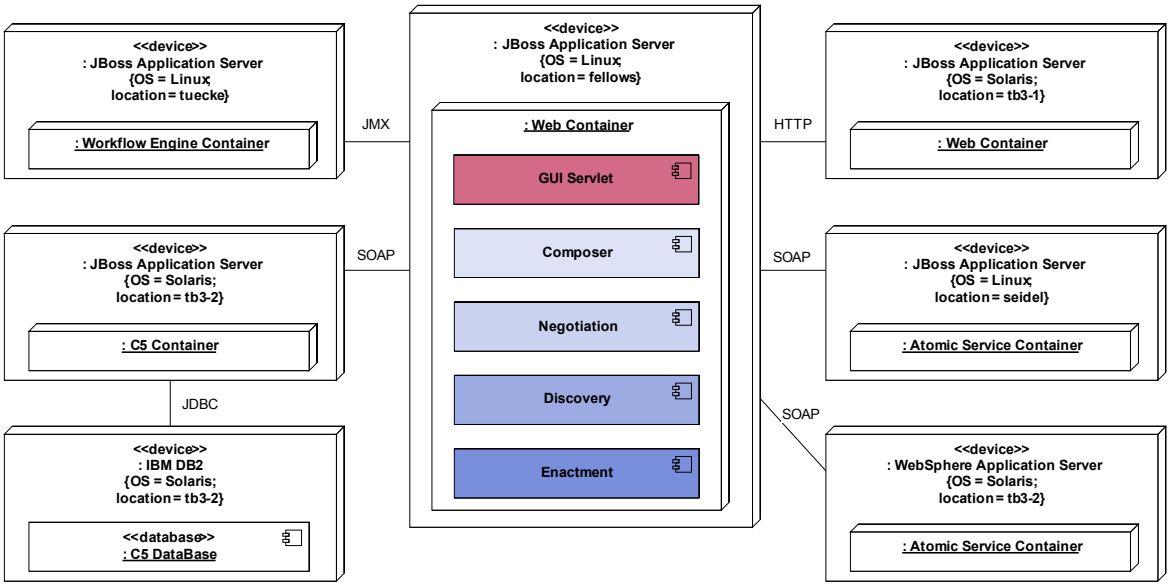

Figure 1: Testbed Infrastructure

Our complex infrastructure for the prototype development, tests, and demonstration is shown in Figure 1. It consists of several Linux and Solaris systems with WebSphere and JBoss as application servers. This heterogeneous environment was chosen to prove the flexibility of the ASG architecture to run both on commercial IBM and open-source products. Up to now the ASG prototypes have been demonstrated solely on open-source products like JBoss Application Server and PostgreSQL. With the migration to commercial products an important step towards productive scenarios is made. In industrial systems support and consulting as well as legal guarantees (for availability, etc.) are essential.

Additionally, J2EE Application Servers of different vendors may be useful to validate the standard compliance of the ASG components and Atomic Services. However, some vendor-specific difficulties occurred during Atomic Service development.

In industrial systems the connection to external services, especially payment service, must be secure and reliable. These security demands are accomplished through encrypted communication and authentication of the partners. The client APIs that we use to interact with the external payment services are based upon Sun-specific encryption algorithms. Since IBM WebSphere Application Server runs on IBM's own JRE, the usage of vendor-specific classes results in "ClassNotFoundException"s.

Hence, our infrastructure contains two hosts for service execution, one runs JBoss AS for the Payment Services requiring Sun JDK and one runs WebSphere AS where all other scenario services are deployed.

At the beginning of the project we set-up a dedicated host for the PXE workflow engine, since it needed an older version of the JBoss application server (4.0.2). Recently, we merged

changes from the newest PXE release with our own modifications. The latest release of PXE supports the most current JBoss Application Server (4.0.3 SP1). However, for system stability and test/debug purposes we left PXE residing on a separate host.

9 CONCLUSION

Our overall goals have been reached within the project. We demonstrated potential productive use in the scenario for dynamic supply chains. To reach this goal, tremendous efforts were required on the implementation of ASG components. Work was required on following components: Composition, Discovery, Negotiation, and Enactment. We inserted into Enactment an open-source workflow engine and hence demonstrated the flexibility of the ASG reference architecture.

Furthermore, we developed a business scenario that is mature for industrial usage. The development process encompassed the following phases: scenario definition, service identification, service landscape definition, service implementation, ontology engineering, and semantic service specification. Separately, a more general methodology describing the application and service engineering lifecycle in ASG context has been evolved. We presented the results at Semantics 2005, Vienna.

For our scenario we also integrated with ASG several external services from providers like PayPal, Saferpay, Directi, Denic, and Verisign. Using test accounts the scenario demonstrate the ordering process of webhosting and domain products – including their payment and the local set-up of websites.

Our reference application according to the specified scenario shows the capability of our prototype to react on a service failure (unavailability) and to renegotiate with another service with similar conditions. This adaptive behaviour increases the overall system's fault tolerance.

Once a process is composed that fulfils a set of requested conditions, subsequent components (Negotiation, Enactment) automatically generates a BPEL compliant process and executes it.