Demonstrating WSMX: Least Cost Supply Management

Eyal Oren[2], Alexander Wahler[1], Bernhard Schreder[1], Aleksandar Balaban[1], Michal Zaremba[2], and Maciej Zaremba[2]

[1] NIWA Web Solutions, Vienna, Austria
`<lastname>@niwa.at`
[2] Digital Enterprise Research Institute (DERI), Galway, Ireland*
`<firstname.lastname>@deri.org`

Abstract Current web service technologies lack semantic descriptions of functionality and requirements; semantic markup of web services would allow interoperability and dynamic discovery of services. The Web Service Modelling Ontology (WSMO) provides a framework for semantically describing web services, ontologies, goals and mediators. WSMX is an execution environment for WSMO allowing discovery, mediation and invocation of semantically described services. We give an overview of the current state of WSMX and demonstrate how WSMX can be used in ordering a broadband Internet line. We note two additional requirements for a web service execution environment: that it should be possible to partially defer web service descriptions until runtime and that it should be possible to execute complex goals. We describe how we augmented the software to support these requirements.

1 Introduction

The emerging web services technology has the potential of offering the industry new ways to serve their customers. For many, the long-term goal of web service effort is seamless inter-operation among networked programs and devices. Once achieved, many see web services as providing the infrastructure for universal plug-and-play and ubiquitous computing. Web services expose functions and information of systems and allow companies to creatively configure existing offerings into new bundles of products and services. Once exposed, web services can easily be accessed and reused, thereby greatly reducing total cost of ownership of service operator systems. Semantically enriching web service descriptions enhances the possibility of discovering and combining services. Enriching web services with semantic descriptions of data enables automatic data exchange and communication with third party suppliers. Organisations should provide their functionalities as web services; semantic technologies would ensure interoperability of these services and allow automated business collaboration.

* This material is based upon works supported by the Science Foundation Ireland under Grant No. 02/CE1/I131.

Business process technologies allow organisational entities to formalise specific processes of their business in order to support automation of these, thereby enhancing production or other business areas. Such technologies promise significant improvements in business solutions. Several techniques for business process representation and management have been developed. At this point in time, around 350 [5, page 4] different business process technologies exist.

The problem currently faced within IT-solutions for business support is the integration problem [2]: techniques are needed for making different resources interoperable, thus supporting automated business collaboration between heterogeneous resources. Another challenge is to make use of the Internet as a world wide information and communication platform for business interaction. Therefore, the web and web services, especially the semantic web along with semantic web services are considered as technologies that can solve the problems currently faced within IT business solutions.

The Web Service Modelling Ontology (WSMO)[3] is a research initiative developing a framework for semantic web services. WSMX[4] is a software system based on WSMO that provides discovery, mediation and invocation of semantic web services. Recently a first milestone of WSMX has been released: the architecture is in place and initial versions of all components have been implemented.

This paper describes how to use WSMX in a process of ordering a broadband Internet line. First we describe this use case in section 2. Then, in section 3, we give an overview of WSMX and explain how WSMX is applied to the use case. In section 4 we note two additional requirements for the software and describe how we extended WSMX to support these; we conclude in section 5.

2 Use Case: Ordering Broadband Internet

For demonstrating the potential of WSMX we selected a use case from the telecommunication sector. Many Internet service providers are extending their businesses with wholesaling of mobile and fixed line telephone services and unbundled data lines. To stay competitive in the future market they need technologies for easy and flexible integration of suppliers that offer these services.

An ADSL line is a broadband Internet connection on top of a regular telephone line. In Austria several suppliers of unbundled ADSL lines are available depending on the region where the customer is located. Most wholesalers need a flexible and dynamic integration of these suppliers in their back-end systems. These systems have to handle the ordering and service requests of the customers and forward them to the selected least cost supplier. The service interfaces of the suppliers (their offered web services for the ADSL ordering process) differ. For example, to check the availability of the ADSL line at some address, supplier A needs both name and address data in one message and does not distinguish between door number and staircase numbers; supplier B however needs only the address data in this first step and requests personal data later.

[3] see `http://www.wsmo.org`

[4] see `http://www.wsmx.org`

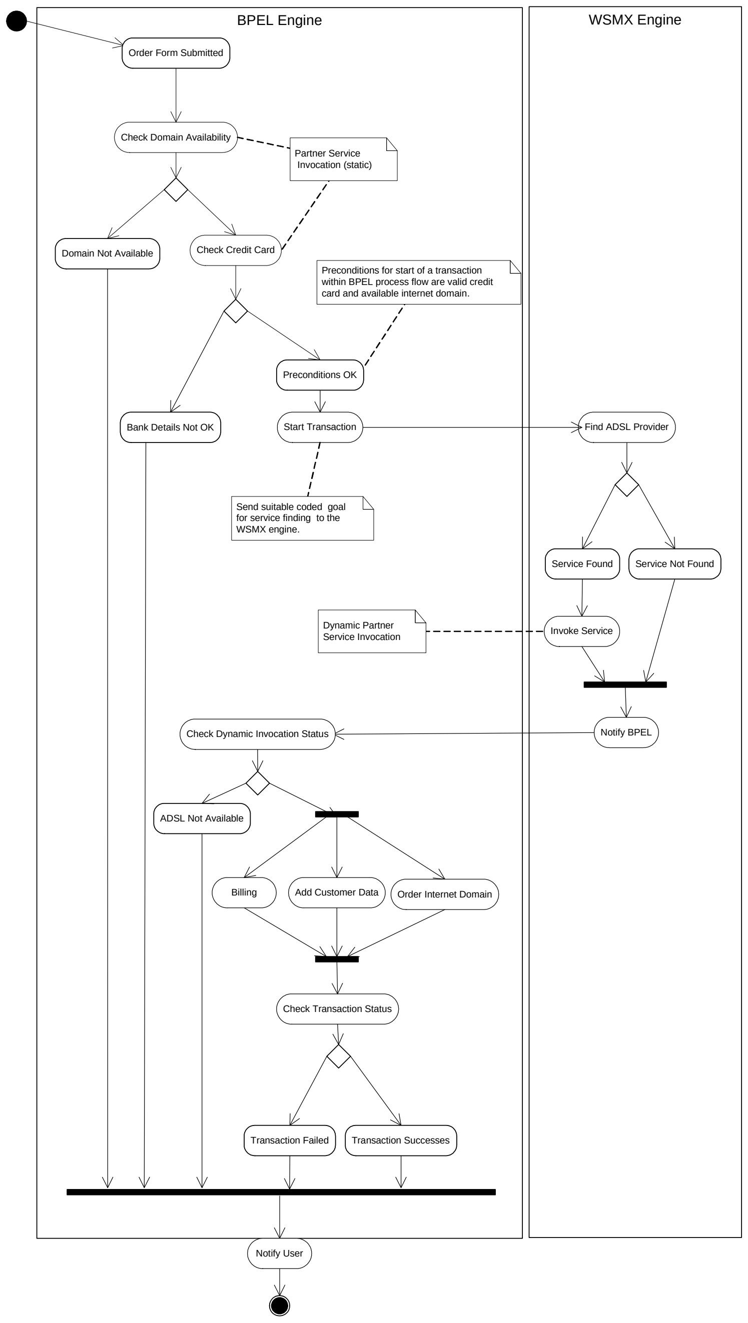

Figure 1. ADSL ordering process

Figure 1 gives an overview of a simplified ADSL ordering process. A new customer activates the process by submitting a completed order form. The information in this order form is used to invoke several external web services. The first step is to check whether the customer-supplied domain name is available and the credit card information is correct. Then the ordering process commences, executing in a transactional scope which allows for the rolling-back of completed steps in case of an error.

The first activity inside this scope is the order of the ADSL line by invoking a web service of an external supplier. Since several suppliers provide the same functionality (with different conditions and costs) one of these suppliers has to be chosen. The dynamic discovery, selection and invocation of these services is delegated to WSMX: the system takes care of finding the least-cost supplier, mediating the customer's data to the ontology of the supplier and invoking the selected supplier. In section 3 we describe how WSMX achieves these tasks.

Following this step several additional operations are necessary to complete the ordering process, including ordering the domain, entering the customer data in the customer database of the provider and billing the customer for the product bundle. The process then responds back to the customer, reporting either the successful order of the whole product bundle or providing information about its failure.

At the moment practical implementations of these processes in Austrian telecommunication providers are hard-wired (supported by emails, faxes and human interactions) because of a lack of automation. We expect WSMX to provide essential technologies for extensive automation of these processes and solutions for the integration problem.

3 WSMX Overview

The WSMX platform aims to offer complete support for interacting with semantic web services. WSMX is based on the framework offered by WSMO, a meta-ontology describing various aspects related to semantic web services. WSMO is based on four concepts: web services, ontologies, goals and mediators.

Web services are units of functionality; every web service has exactly one capability, that describes logically what this web service can offer. A capability of a web service consists of preconditions, postconditions, assumptions and effects. The capability of a web service is interpreted as follows: the service guarantees that, if at execution time of the service the preconditions and assumptions hold, then the postconditions and effects will hold after execution. A web service has a number of interfaces, which specify how to communicate with it.

Goals describe some state that a user may want to achieve. *Ontologies* are the formal specification of the knowledge domain used by both the web service to express its capability, and by the goal to express the desired world state. *Mediators* are used to solve different interoperability problems, for example differences in the ontologies used by a web service and a goal.

WSMO is based on the notion of reuse of components and maximal decoupling between concepts: web services and goals are strongly decoupled; mediators enable reuse of components by refining or adapting an existing component. This approach is similar to problem solving methods, c.f. [4,3].

WSMX is a software system that operates on these concepts. WSMX manages a repository of web services, ontologies and mediators – all of whom are semantically described and fed into WSMX. WSMX can achieve a user's goal by dynamically selecting a matching web service, mediating the data that needs to be communicated to this service and invoking it. A simplified view of the architecture that focuses on the relevant parts is shown in figure 2.

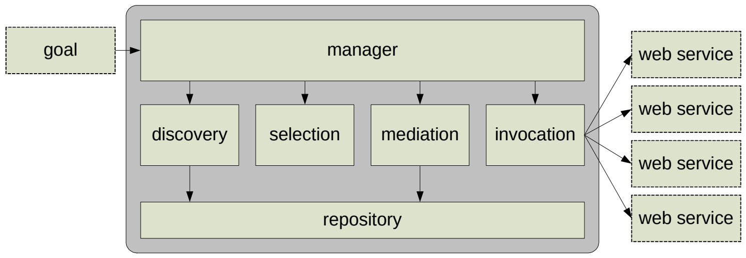

Figure 2. Simplified WSMX Architecture

In this use case WSMX takes care of ordering the ADSL line from a provider: WSMX is asked to achieve the goal "ADSL line for the supplied phone number". WSMX performs dynamic discovery of all providers capable of delivering ADSL lines, selection of the least-cost supplier, mediation of the customer's data to the format of the supplier and invocation of the selected supplier with this mediated data.

The *discovery* component is responsible for finding all services that are able to provide ADSL lines. As we will explain in section 4, we extended WSMX to be able to check at runtime for exactly which numbers the suppliers can provide ADSL lines. The *selection* component then selects from the discovered services the one that is most preferable, based on the conditions and prices of the different providers.

If the goal and the selected web service are specified using different ontologies (which is most likely) the *mediation* component needs to solve this. This component uses specified mappings between ontologies in order to mediate the payload data from the format of the customer to the format that is used by the selected web service. The specification of these mapping rules is done semi-automatically, the mediation itself is carried out automatically (mediation is only possible if mapping rules between these two ontologies have been specified). The *invocation* component then invokes the selected web service using the mediated payload data.

An important business element in the context of dynamic discovery is the notion of trust, in two aspects. First of all a business may not completely trust the outcome of the discovery process: is this really the best provider for their goal. Secondly, a business might not want to use the discovered provider without a prior established relation: they might not trust arbitrary partners. We do not address these issues here; these have to be addressed in other work.

4 Extending WSMX

The use case shows that for WSMX to be useful in real-world scenarios a number of things are necessary. Firstly the assumption of WSMX –that providers are able to completely describe the functionality of their web services at design time– is unreasonable. The exact functionality of services can be rapidly changing; providers need a mechanism to defer specification of their capabilities until runtime.

Secondly, WSMX has a quite limited operational behaviour: WSMX can match services against a goal and then select one of those matching services for invocation. However, in real-world scenarios richer process models are necessary to guide the software. Therefore it would be useful if WSMX could understand some process modelling language, so that complex goals could be specified consisting of several subgoals that should be achieved in a certain order.

We will explain and address these two concerns in the following sections.

4.1 Conditional Web Services

An important element of this use case is that providers are not really able to fully specify their capabilities upfront. All providers state they can provide ADSL lines for telephone numbers, but the exact set of numbers for which they can provide a line is changing so rapidly that they cannot describe this range in their capability. The set of numbers for which each provider is responsible changes very frequently and it is not feasible to continuously update their capability descriptions.

The question is how to correctly specify the capability of a provider without explicitly stating the range of telephone numbers. This can be achieved by introducing a built-in function in the logical expression that will at runtime evaluate to a web service invocation. This web service invocation takes as input the actual phone number of the customer and returns whether the provider supports an ADSL connection for that number *at that specific moment.* All providers have to offer this web service, which indeed they do in reality.

For technical reasons we chose to implement a slightly different solution that is less generic than the above-mentioned one but easier to implement. In future work we would like to further explore the above-mentioned solution. Instead of extending the logical language to include the possibility of defining functions that are implemented by a web service, we decided to extend the capabilities of web services with what we call a *conditional web service.* The capability of a web service should now be interpreted as follows: the service guarantees that, if at execution time of the service the preconditions and assumptions hold *and the conditional web service evaluates to "true"*, then the postconditions and effects will hold after execution.

Introducing this concept of a *conditional web service* changes the operational behaviour of WSMX. Let us say some web service W matches a goal G and the capability of this web service contains a conditional web service C. To decide whether the capability of W is valid and thus whether this service really matches the goal, WSMX needs to execute the conditional web service C.

However, executing this conditional web service is not straightforward: it may well be that mediation is needed because the conditional web service is using a different ontology than the goal. Ideally we would like to reuse WSMX for this complicated procedure and to send this subgoal (invoking the conditional web service C with some payload) through the WSMX system. However, the web service that should be used for this subgoal is known[5] so matchmaking and selection should be skipped. This means that for executing the conditional web service a different operational behaviour is needed than for executing a normal goal: some components can be skipped. To make this scenario technically possible we need to make some adjustments to the software.

In the WSMX architecture all communication between components consists of events. In order to enable the execution of conditional web services, and to use the outcome of this execution in the execution of the parent service, we need a mechanism to relate different events. We take the following approach:

For all conditional web services a semantic description is available in the WSMX repository, otherwise we do not know which ontology it uses or how to invoke it. If a conditional web service is part of the capability of a web service then it must be executed during matchmaking, to check whether a capability actually holds. Each conditional web service creates a new event in the system (a sub-event of the main event), which does not follow the same execution semantics as the main event. Instead, a sub-event only needs mediation and invocation, but we do not need to discover and select the web service to achieve the sub-event.

All sub-events return typed values; if all return values from conditional web services are "true" values then the conditional web service is fulfilled and execution of the main web service can be performed. A new component, the *event correlation manager*, maintains the relationships between the main event and the sub-events. When a conditional web service is encountered, the correlation manager puts the main event into a sleep state and initialises the sub-events; when all the sub-events reach a final state the correlation manager wakes up the main event to process the result of the conditional web service.

When a conditional web service is defined in a web service capability the *event correlation manager* takes over, puts the running event into sleep and generates new sub-events that are executed using different execution semantics.

As depicted in figure 3 sub-events can be nested recursively. A newly created sub-event can be made to skip certain components, by giving it a specific initial

[5] it is the service C, as specified in the capability of WS

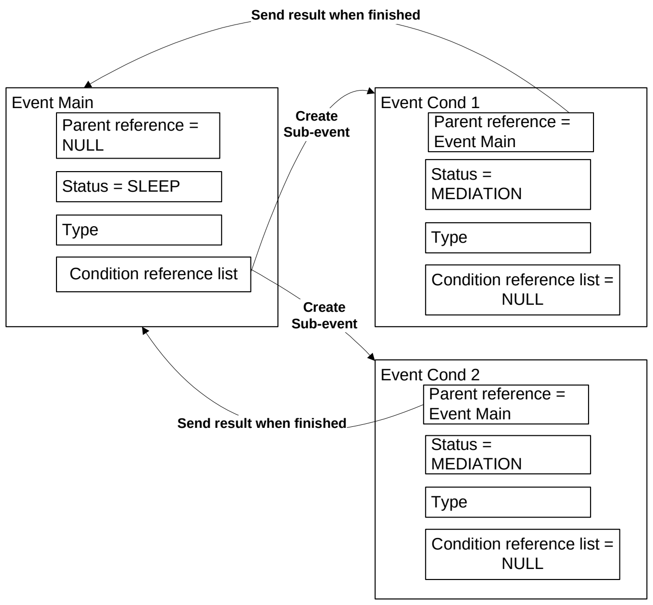

Figure 3. Example of relation between main event and sub-events

state; to skip matchmaking for example, a new sub-event could be given the `before_mediation` status, which will make only mediation and invocation take place.

These conditional web service invocations can be recursive, i.e. the web service we execute to determine a condition can itself specify a conditional web service. When trying to execute these recursions we may encounter infinite recursions and execute in endless loops without reaching a service that returns an answer. We have not yet addressed these issues, but we envisage recursion-detection mechanisms in the software to prevent us from being caught in such loops.

4.2 Business Process Engine

In the presented use case WSMX is only used in a certain part of the overall process. In figure 1 the whole ordering process is shown: WSMX is only used to order the line, but the rest of the process is executed by a process execution engine[6]. The process execution engine is able to understand the given specification of the ordering process and manages the runtime execution of this process. It keeps track of the process and tells components to execute a certain task.

It would be very useful if WSMX could understand such a process specification. One could specify a goal that consists of this process specification and

[6] of course one could also hard-code the process into a computer program; that does not change our argument

WSMX would, while executing the goal, manage the process and execute the subgoals in it. Each atomic task inside the process model would be executed through WSMX, using the powerful matchmaking, mediation and invocation capabilities. The WSMO working group is committed to include specification of complex goals in WSMO. Once such a specification is made WSMX will support execution of these complex goals by including a process management component.

However, no decision has yet been made on a formalism for specifying complex goals; in the meantime a temporary solution would be to embed WSMX inside a business process engine[7]. This solution could be used with any web services business process engine, for instance YAWL [1] or WS-BPEL[8]. We demonstrate this simple idea with WS-BPEL – our approach is however as applicable to any other business process language.

WS-BPEL is a language for modelling business processes based on web services; it is widely used for automation of business processes by execution of the modelled processes. A WS-BPEL process consists of basic activities, that are composed into complex activities using routing constructs. The basic activities are essentially sending and receiving data to and from other web services. The exact paths followed during execution can depend on data gathered during execution.

The web services with which a WS-BPEL process can communicate have to be known at design time. This is clearly not a desired situation – the specification itself states that mechanisms for dynamic service discovery should be available. However, no such mechanism is implemented in the current WS-BPEL specification. WSMX on the other hand is able to perform this dynamic discovery and to mediate the payload to a format that the selected web service can understand.

The integration is straightforward: given a certain WS-BPEL process, one changes in all atomic tasks the web service calls $W_1..W_n$ into calls $W'_1..W'_n$. Each call W_i is an invocation of a specific web service, the replacing W'_i is an invocation of WSMX asking to achieve goal G_i. This goal G_i corresponds to what web service W_i would have achieved, but by using WSMX to achieve this goal the actual service that is used is dynamically selected during runtime. The challenge is to find an appropriate semantic description of the goal such that it corresponds with what the web service W_i would have achieved in the original process.

A major drawback of WS-BPEL is the need to statically define the used web services at design time; it is not possible to communicate with services that were unknown at design time. WSMX is however well able to dynamically discover new services that provide some wanted functionality: an integration of these two approaches leads to a process execution engine that can execute some complex business process model and, during execution, dynamically discover relevant services.

[7] which could be seen as the "opposite" approach

[8] formerly known as BPEL4WS, see `http://www-106.ibm.com/developerworks/library/ws-bpel/`

5 Conclusion and Future Work

We have discussed a use case that deals with ordering broadband Internet connections from several providers. This scenario is currently implemented without much automation – using human interaction and human communication channels. Information technology solutions face the integration problem: to integrate heterogeneous resources and heterogeneous communication channels.

We have presented WSMX, an execution environment for semantic web services that is able to perform dynamic discovery, mediation and invocation of services; WSMX provides the technology to support this use case. The use case however also makes clear that the general assumption of WSMX –that providers should completely describe the functionality of their services at design time– is unreasonable. We have presented a general solution to this problem and shown how we have technically implemented this solution.

We have also argued for a business process management component as part of WSMX. As a temporary solution we have shown how to integrate the widely used process modelling language WS-BPEL with WSMX. This combination provides a process modelling engine that is able to execute complex processes and can dynamically discover services to solve atomic tasks, including mediation of data into a format understandable by the selected service.

We hope to address in future work the two shortcomings noted in this paper: firstly, to extend the logical language to include a built-in function that evaluates during runtime to a web service call; secondly, to include a process modelling component in WSMX that is able to execute complex goals.

Acknowledgements We would like to thank the referees for their valuable comments on a previous version of this article.

References

1. W. M. P. v. d. Aalst and A. t. Hofstede. YAWL: Yet another workflow language. Technical Report FIT-TR-2003-04, Queensland University of Technology, 2003.
2. C. Bussler. *B2B Integration, Concepts and Architecture.* Springer-Verlag, 2003.
3. D. Fensel. Problem-solving methods: Understanding, development, description, and reuse. In *Lecture Notes On Artificial Intelligence*, number 1791. Springer, 2000.
4. D. Fensel et al. The unified problem-solving method development language UPML. *Knowl. Inf. Syst.*, 5(1):83–131, 2003.
5. L. Hommes. *The Evaluation of Business Process Modeling Techniques.* PhD thesis, Delft University of Technology, 2004.