WSMX: A Semantic Service Oriented Middleware for B2B Integration

WSMX: A Semantic Service Oriented Middleware for B2B Integration*

Thomas Haselwanter[1], Paavo Kotinurmi[1,2], Matthew Moran[1], Tomas Vitvar[1], and Maciej Zaremba[1]

[1] Digital Enterprise Research Institute
University of Innsbruck, Austria
National University of Ireland in Galway, Ireland
firstname.lastname@deri.org
[2] Helsinki University of Technology, Finland

Abstract In this paper we present a B2B integration scenario building on the principles of Semantic Web services. For this scenario we show the benefits of semantic descriptions used within the integration process to enable conversation between systems with data and process mediation of services. We illustrate our approach on the WSMX – a middleware platform built specifically to enact semantic service oriented architectures.

1 Introduction

Although Business-to-Business (B2B) standards such as RosettaNet, EDI and ebXML have brought new value to inter-enterprise integration, they still suffer from two drawbacks. All partners must agree to use the same standard and often the rigid configuration of standards makes them difficult to reconfigure, reuse and maintain. In order to overcome some of the difficulties of B2B integration, semantic technologies offer a promising potential to enable more flexible integration that is more adaptive to changes that might occur over a software system's lifetime [4]. There remains, however, very few realistic publicly implemented scenarios demonstrating the benefits of this technology. In this respect, we aim to showcase how Semantic Web service (SWS) technology can be used to facilitate the dynamics for B2B integration. We base our work on specifications of WSMO[5], WSML[5] and WSMX[3] providing a conceptual framework, ontology language and execution environment for Semantic Web services. In addition, we make use of the RosettaNet B2B standard – an industry standard providing definition of inter-company choreographies (e.g. PIP3A4 Request Purchase Order (PO)) as well as structure and semantics for business messages. In this paper we show how these technologies are used in a real-world scenario involving (1) semantic representation of XML schema for RosettaNet as well as a proprietary purchase order using the WSML ontology language, (2) semantic representation of services provided by partners using WSMO, (3) executing a conversation

* *This work is supported by the Science Foundation Ireland Grant No. SFI/02/CE1/I131, and the EU projects Knowledge Web (FP6-507482), DIP (FP6-507483) and ASG (FP6-C004617).*

between partner services using the WSMX integration middleware, and (4) application of data and process mediation between heterogeneous services where necessary.

2 Architecture

In figure 1, a fictitious trading company called Moon uses two back-end systems to manage its order processing, namely, a Customer Relationship Management system (CRM) and an Order Management system (OMS). Moon has signed agreements to exchange purchase order messages with a partner company called Blue using the RosettaNet PIP 3A4. We use SWS technology to facilitate conversation between all systems, to mediate between the PIP 3A4 and the XML schema used by Moon, and to ensure that the message exchange between both parties is correctly choreographed. Following is a description of the basic blocks of the architecture.

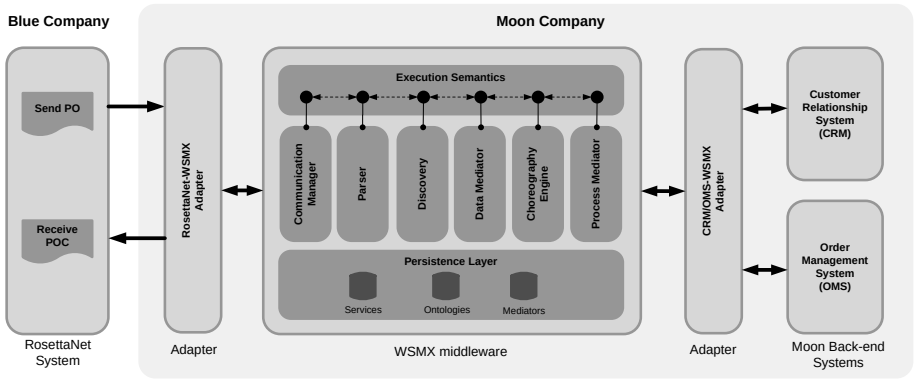

Figure 1. Architecture Overview

- **Existing Systems.** The existing systems are Moon's back-end applications including CRM and OMS systems as well as Blue's RosettaNet system. Each system communicates using different formats, i.e. Blue's RosettaNet system communicates according to the RosettaNet PIP 3A4 PO, whereas communication with the CRM and OMS systems is proprietary, specified in their WSDL descriptions.
- **Adapters.** Since WSMX internally operates on the semantic level (WSML), adapters facilitate *lifting* and *lowering* operations to transform between XML and WSML. In addition, it also identifies the WSMO Goal that corresponds to a PO request and sends it to WSMX.
- **WSMX.** WSMX is the integration platform which facilitates the integration process between different systems. The integration process is defined by the WSMX execution semantics, i.e. interactions of middleware services including discovery, mediation, invocation, choreography, repository, etc.

There are two phases to the integration of the Blue and Moon partners: (1) *integration setup phase* and (2) *integration runtime phase*. During the setup

phase, the development of adapters, WSMO ontologies and services, rules for lifting/lowering, mapping rules between ontologies are carried out for RosettaNet, OMS and CRM systems. The focus of this paper is on the runtime phase describing interactions between Blue and Moon systems.

2.1 Activity Diagram

In this section we describe interactions between the Blue and Moon systems facilitated by WSMX through its middleware services as depicted in the figure 2. We refer to parts of the figure using numbers before title of each subsection.

1 – Sending Request. A PIP3A4 PO message is sent from the RosettaNet system to the entry point of the RosettaNet-WSMX adapter. On successful reception of the message by the adapter, an acknowledgment is sent back to the RosettaNet system. In the RosettaNet-WSMX adapter, the PO XML message is lifted to WSML according to the PIP3A4 ontology and rules for lifting using XSLT. Finally, a WSMO Goal is created from the PO message including the definition of the desired capability and a choreography. The capability of the requester (Blue company) is used during the discovery process whereas the Goal choreography describes how the requester wishes to interact with the environment. After the goal is created, it is sent to WSMX through the *AchieveGoal* entrypoint. In return, a *context* is received containing the identification of the *conversation* – this information is used in subsequent asynchronous calls from the requester.

2 – Discovery and Conversation Setup. The *AchieveGoal* entrypoint is implemented by the WSMX Communication Manager – the WSMX middleware service, which facilitates the inbound and outbound communication with the WSMX environment. After receipt of the goal, the Communication Manager initiates the *execution semantics* which manages the whole integration process. The Communication Manager sends the WSML goal to the instance of the execution semantics, which in turn invokes the WSMX Parser returning the Goal parsed into an internal system object. The next step is to invoke the discovery middleware service in order to match the requested capability of the Goal with the capabilities of services registered in the WSMX repository. Since we do not concentrate on the discovery in this paper, we use a very simplified approach when only one service in the repository (CRM/OMS service) can be matched with the goal. After discovery, the execution semantics registers both the requester's and the provider's choreography with the Choreography Engine (these choreographies are part of the goal and service descriptions respectively). Both choreographies are set to a state where they wait for incoming messages that could fire a transition rule. This completes the conversation setup.

3 – Conversation with Requester. The instance data for the goal is sent from the RosettaNet-WSMX adapter to the WSMX asynchronously by invoking the *receiveData* entrypoint. The data in WSML (WSMLmsg) is passed through the Communication Manager to the execution semantics, they are parsed and sent to the WSMX Process Mediator. The first task of the WSMX Process Mediator is to

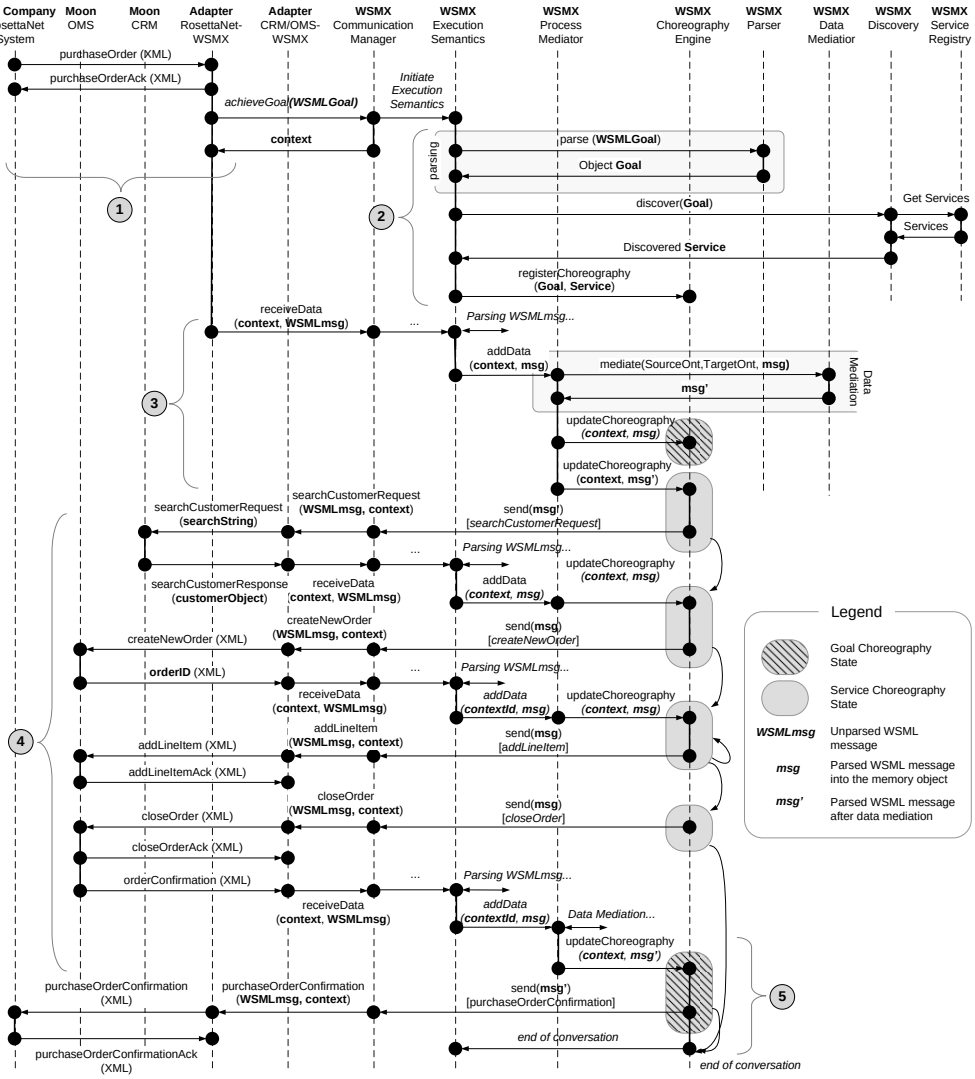

Figure 2. Activity Diagram

decide, which data will be added to requester's or provider's choreography[3] – this decision is based on analysis of both choreographies and concepts used by these choreographies. Process Mediator first updates the memory of the requester's choreography with the information that the PO request has been sent. The Process Mediator then evaluates how data should be added to the memory of the provider's choreography – data must be first mediated to the ontology used by the provider. For this purpose, the source ontology of the requester, target ontology of the provider and the instance data are passed to the WSMX Data Mediator. Data mediation is performed by execution of mapping rules between both ontologies (these mapping rules are stored within WSMX and have been created and registered during the integration setup phase). Once mediation is complete, the mediated data is added to the provider's choreography.

4 – Conversation with Provider (opening order, add line items, closing order). Once the requester's and provider's choreographies have been updated, the Choreography Engine processes each to evaluate if any transition rules could be fired. The requester's choreography remains in the waiting state as no rule can be evaluated at this stage. For the provider's choreography, the Choreography Engine finds the rule shown in the listing 1.1 (lines 14-21). Here, the Choreography Engine matches the data in the memory with the the antecedent of the rule and performs the action of the rule's consequent. The rule says that the message *SearchCustomerRequest* with data *searchString* should be sent to the service provider (this data has been previously added to the choreography memory after the mediation - here, *searchString* corresponds to the *customerId* from the requester's ontology). The Choreography Engine then waits for the *SearchCustomerResponse* message to be sent as a response from the provider. Sending the message to the service provider is initiated by Choreography Engine passing the message to the Communication Manager which, according to the grounding defined in the choreography, passes the message to the *searchCustomer* entrypoint of the CRM/OMS-WSMX Adapter. In the adapter, lowering of the WSML message to XML is performed using the lowering rules for the CRM/OMS ontology and the CRM XML Schema. After that, the actual service of the CRM system behind the adapter is invoked, passing the parameter of the *searchString*. The CRM system returns back to the CRM/OMS Adapter a resulting *customerObject* captured in XML. The XML data is lifted to the CRM/OMS ontology, passed to the WSMX, parsed, evaluated by the WSMX Process Mediator and added to the provider's choreography memory. Once the memory of the provider's choreography is updated, the next rule is evaluated resulting in sending a *createNewOrder* message to the Moon OMS system. This process is analogous to one described before. As a result, the *orderID* sent out from the OMS system is again added to the memory of the provider's choreography. After the order is created (opened) in the OMS system, the individual items to be ordered need to be added to that order. These items were previously sent in one message as part of order request from Blue's RosettaNet system (i.e. a collection of *ProductLineItem*) which must be now sent to the OMS system

[3] Choreographies of WSMO services are modeled as Abstract State Machines [2]

individually. As part of the data mediation in the step 3, the collection of items from the RosettaNet order request have been split into individual items which format is described by the provider's ontology. At that stage, the Process Mediator also added these items into the provider's choreography. The next rule to be evaluated now is the rule of sending *addLineItem* message with data of one *lineItem* from the choreography memory. Since there is more then one line item in the memory, this rule will be evaluated several items until all line items from the ontology have been sent to the OMS system. When all line items have been sent, the next rule is to close the order in the OMS system. The *closeOrder* message is sent out from the WSMX to the OMS system and since no additional rules from the provider's choreography can be evaluated, the choreography gets to the end of conversation state.

The listing 1.1 shows the fragment of the provider's choreography and the first selected rule from the requester's choreography described above. The choreography is described from the service point of view thus the rule says that the service expects to receive the *SearchCustomerRequest* message and send the reply *SearchCustomerResponse* message. The choreography is part of the service definition which in addition also contains definition of *non-functional properties* and *capability*. For brevity, these elements are not included in the listing.

```
1   ...
2   choreography MoonWSChoreography
3       stateSignature _"http://www.example.org/ontologies/sws-challenge/MoonWS#statesignature"
4         importsOntology {_"http://www.example.org/ontologies/sws-challenge/Moon",
5                          _"http://www.example.org/ontologies/choreographyOnto" }
6
7         in moon#SearchCustomerRequest withGrounding { _"http://intranet.moon.local/wsmx/services/
                CRMOMSAdapter?WSDL#wsdl.interfaceMessageReference(CRMOMSAdapter/CRMsearch/
                in0)"}
8         ...
9         out moon#SearchCustomerResponse
10        ...
11        controlled oasm#ControlState
12
13  transitionRules _"http://www.example.org/ontologies/sws-challenge/MoonWS#transitionRules"
14    forall {?controlstate, ?request} with (
15        ?controlstate[oasm#value hasValue oasm#InitialState] memberOf oasm#ControlState and
16        ?request memberOf moon#SearchCustomerRequest
17      ) do
18        add(?controlstate[oasm#value hasValue moonc#SearchCustomer])
19        delete(?controlstate[oasm#value hasValue oasm#InitialState])
20        add(_# memberOf moon#SearchCustomerResponse)
21    endForall
22    ...
```

Listing 1.1. Requester's Service Choreography

5 – Conversation with Requester (order confirmation, end of conversation). When the order in OMS system is closed, the OMS system replies with *orderConfirmation*. After lifting and parsing of the message, the Process Mediator first invokes the mediation of the data to the requester's ontology and then adds the data to the memory of the requester's choreography. The next rule of the requester's choreography can be then evaluated saying that *purchaseOrderConfirmation* message needs to be sent to the RosettaNet system. After the

message is sent, no additional rules can be evaluated from the requester's choreography, thus the choreography gets to the end of conversation state. Since both requester's and provider's choreography are in the state of end of conversation, the Choreography Engine notifies the execution semantics and the conversation is closed.

3 Conclusion and Future Work

This work addresses the fact that although research into the area of Semantic Web services is well established, there is a scarcity of implemented use cases demonstrating the potential benefits. Existing approaches to dynamic or semantic B2B integration such as [1], [4], [6] are mostly conceptual with lack of demonstration and evaluation of real-world case scenarios. The system presented here has been implemented according to the scenario from the SWS Challenge[4] addressing data and process heterogeneities in a B2B integration. It has been evaluated how our solution adapts to changes of back-end systems with needs to change the execution code (success level 1), data/configuration of the system (success level 2), no changes in code and data (success level 3)[5]. For data mediation we had to make some changes in code due to forced limitations of existing data mediation tools. For process mediation we changed only description of service interfaces (choreographies) according to the changes in back-end systems. Ultimately, we aim to the level when our system adapts without changes in code and configuration – the adaptation will be purely based on reasoning over semantic descriptions of services, their information models and interfaces. We also plan to expand our solution to cover more B2B standards and to integrate key enterprise infrastructure systems such as policy management, service quality assurance, etc.

References

1. N. Anicic, N. Ivezic, and A. Jones. An Architecture for Semantic Enterprise Application Integration Standards. In *Interoperability of Enterprise Software and Applications*, pp. 25–34. Springer, 2006.
2. E. Brger and R. Strk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag, 2003.
3. A. Haller, *et al.* WSMX – A Semantic Service-Oriented Architecture. In *Proc. of the 3rd Int. Conf. on Web Services*, pp. 321 – 328. IEEE Computer Society, 2005.
4. C. Preist, *et al.* Automated Business-to-Business Integration of a Logistics Supply Chain using Semantic Web Services Technology. In *Proc. of 4th Int. Semantic Web Conference*. 2005.
5. D. Roman, *et al.* Web Service Modeling Ontology. *Applied Ontologies*, 1(1):77 – 106, 2005.
6. K. Verma, *et al.* The METEOR-S Approach for Configuring and Executing Dynamic Web Processes, available at http://lsdis.cs.uga.edu/projects/meteor-s/techRep6-24-05.pdf. Tech. rep., 2005.

[4] http://www.sws-challenge.org
[5] http://sws-challenge.org/wiki/index.php/Workshop_Budva