

# Data and Process Mediation Support for B2B Integration

Maciej Zaremba<sup>1</sup>, Maximilian Herold<sup>1</sup>, Raluca Zaharia<sup>1</sup>, and Tomas Vitvar<sup>2</sup>

<sup>1</sup> Digital Enterprise Research Institute  
National University of Ireland in Galway, Ireland  
`firstname.lastname@deri.org`

<sup>2</sup> Semantic Technology Institute (STI2)  
University of Innsbruck, Austria  
`tomas.vitvar@sti2.at`

**Abstract** In this paper we present how Semantic Web Service technology can be used to overcome process and data heterogeneity in a B2B integration scenario. While one partner uses standards like RosettaNet for product purchase and UNIFI ISO 20022 for electronic payments in its message exchange process and message definition, the other one operates on non-standard proprietary solution based on a combination of WSDL and XML Schema. For this scenario we show the benefits of semantic descriptions which are used within the integration process to enable rule-based data and process mediation of services. We illustrate this dynamic integration process on the WSMX – a middleware platform conforming to the principles of a Semantic Service Oriented Architecture.

## 1 Introduction

Inter-enterprise integration is the essential requirement for today's successful business. While technologies around RosettaNet, EDI or ebXML certainly brought new value to inter-enterprise integration, its rigid and hard-wired configuration makes it still difficult to reconfigure, reuse and maintain. In addition, cooperating partners often use different Business-to-Business (B2B) standards thus either adoption of a standard used by a "stronger" partner or maintaining more than one B2B standards within one B2B integration is required.

Semantic technologies offer promising potential to enable B2B integration that is more flexible and adaptive to changes that occur over a software system's lifetime [8]. Semantic Web services (SWS), by augmenting services with semantic descriptions, is one of the candidate technology for more automated and dynamic service integration and discovery. Semantically annotated services promote the integration process by enabling runtime data and process mediation. The scenario used in this paper is based on the requirements for the SWS Challenge<sup>3</sup>, and in particular on a scenario for data and process mediation. In

---

<sup>3</sup> <http://sws-challenge.org>

comparison to the previously developed mediation solution [3,4] we have provided a support for a new electronic payment scenario and our solution has been extended with a fully-fledged rule-based data mediation [6].

In order to address the SWS challenge requirements, we base our solution on the specifications of WSMO[9], WSML[9] and WSMX[12] providing a conceptual framework, ontology language and architecture for Semantic Web Services. The overall contribution of our work is to show: (1) how flat XML schema of RosettaNet, UNIFI ISO 20022, and other messaging schema used by different partners can be semantically enriched using the WSML ontology language, (2) how services provided by partners could be semantically described as WSMO services and built on top of existing systems, (3) how conversation between partners and their services can be facilitated by the WSMX integration middleware enabling semantic integration, and (4) how generic, rule-based data and process mediation can be applied between heterogeneous services within the integration process.

## 2 Solution Architecture

In SWS-Challenge mediation scenario there are two business partners (Moon and Blue) involved that need to have their systems integrated using semantically-enabled technology. The scenario describes how Moon has signed agreements to exchange purchase order messages with its client company called Blue using the RosettaNet PIP 3A4 specification. Details of our solution to RosettaNet scenario has been previously described in [3].

In this paper we provide a support for the new payment scenario<sup>4</sup> and we extend our previous solution with a support for fully-fledged rule-based data mediation. We have resolved technical problems with integrating rule-based Data Mediation component and we showcase a generic data mediation solution. We build our solutions on the SWS framework based on the WSMO (Web Service Modeling Ontology [9]) conceptual model, WSML (Web Service Modeling Language [9]) language for service modeling, WSMX (Web Service Execution Environment[12]) middleware system, and WSMT (Web Service Modelling Toolkit[5]) modelling framework. In order to model the scenario, we use WSMO for modeling of services and goals (i.e. required and offered capabilities) as well as ontologies (i.e. information models on which services and goals are defined) all expressed in the WSML-Flight ontology language. WSML-Flight provides a Datalog expressivity extended with inequality and stratified negation that is sufficient for addressing requirements of SWS Challenge scenarios. We use KAON2 reasoner<sup>5</sup> and IRIS<sup>6</sup> for the inference over WSML-Flight ontologies.

---

<sup>4</sup> [http://sws-challenge.org/wiki/index.php/Scenario:\\_Payment\\_Problem](http://sws-challenge.org/wiki/index.php/Scenario:_Payment_Problem)

<sup>5</sup> <http://kaon2.semanticweb.org>

<sup>6</sup> <http://iris-reasoner.org>

## 2.1 Payment Scenario

In Figure 1, the global architecture of our solution for the case scenario is depicted. The whole integration process of the Blue and Moon companies happens in two phases: (1) *integration setup phase* and (2) *integration runtime phase*. During the setup phase, the integration ontologies are designed including the models used in UNIFI ISO 20022 payment information and the models used by Moon’s financial systems. The design and implementation of adapters, creation of WSMO ontologies and services, rules for lifting/lowering, mapping statements between used ontologies and registration of ontologies, services and mapping statements with WSMX are also carried out. During the runtime phase, interactions between Blue and Moon systems are executed.

In order to address integration of Blue and Moon companies, our goal is to use Semantic Web service technology to facilitate conversation between all systems, to mediate between the data used by Moon, as well as to ensure that the message exchange between all parties is correctly choreographed. Data Mediation was not necessary in payment scenario as most of the integration process was carried out on the Blue side where homogenous ontology of UNIFI ISO 20022 electronic payment information has been used. Due to the simplicity of Moon’s data in this scenario there was no need to provide data mediation support and bank account details provided by Blue service has been directly utilized in integration process. Process mediation is involved in mapping of message exchanges defined by UNIFI ISO 20022 process to those defined in the WSDL of the Moon back-end systems. Conversation between systems including data and process mediation operates on semantic descriptions of messages, thus transformation from messages used by existing systems to ontological level is first performed.

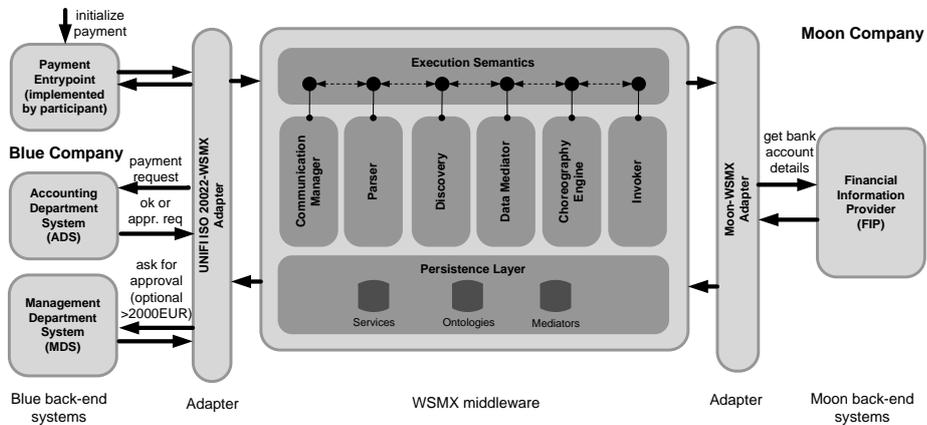


Figure 1. Global Integration Architecture

In our solution, we built the integration between the Blue and Moon systems on the WSMX platform which resides between Moon and Blue infrastructure al-

lowing the seamless integration of all involved systems. XML messages between the partners are lifted to semantically-enabled WSML level. Blue's message initiating B2B interaction is translated into WSMO Goal what allows for goal-driven discovery, service execution and mediation that is provided by WSMX environment. Goals describe requirements over the service to be discovered and are specified independently from the actual service. The following basic blocks are involved in our solution to SWS-Challenge B2B integration:

- **Existing Systems.** The existing systems are Moon's back-end service Financial Information Provider as well as Blue's UNIFI ISO 20022 payment system with Accounting Department System and Management Department System. Each system communicates using different formats, e.g. Blue's systems communicates according to the UNIFI ISO 20022 messages (Payment Information), whereas communication with the Moon's system is more proprietary - specified in their WSDL descriptions. Detail descriptions of these WSDL interfaces can be found at SWS challenge web site.
- **Adapters.** In order to connect existing systems with WSMX, adapters are used to mediate between the different communication protocols and languages. Since WSMX internally operates on the semantic level handling messages in WSML, adapters facilitate lifting and lowering operations allowing message to be transformed from XML to WSML and vice-versa. The adapter also handles the application logic of identifying a valid Goal to be sent to the WSMX for the incoming message and subsequently sending the lifted form (WSML) of the purchase order message. Goal-based invocation is the basis for advanced semantic discovery and mediation. In Figure 1, the UNIFI ISO 20022-WSMX and Moon-WSMX adapters are used for connection to the Blue and the Moon system.
- **WSMX.** WSMX is the integration platform which facilitates the integration process between different systems. The integration process is defined by the execution semantics describing interactions of middleware services including discovery, mediation, invocation, choreography, repository services, etc. Detail descriptions of execution semantics and middleware services for our use case is given later in this section.

A payment request is sent from the client in XML to the entry point of UNIFI ISO 20022-WSMX adapter. In the UNIFI ISO 20022-WSMX adapter, the message captured in XML is lifted to WSML according to the UNIFI ISO 20022 ontology and rules for lifting. The abstract WSMO goal<sup>7</sup> is created including definitions of requested capabilities and a choreography. Requested capabilities describe the desired capability of the requester (Blue company) used during the discovery process whereas goal choreography describes how the requester wishes to interact with the environment. Since a WSMO service is, from the WSMX point of view, represented by an adapter (the adapter can be understood as a wrapper around existing application – in our case Blue's RosettaNet system), the choreography here reflects the communication pattern of the adapter (hence

---

<sup>7</sup> We refer to the abstract goal as a goal which contains no instance data (input values)

it does not include interactions regarding acknowledgments of messages). After the goal is created, it is sent as a WSML message to the WSMX environment through the *AchieveGoal* endpoint.

The WSML message is passed through the Communication Manager to the execution semantics which again first parses the data into the memory object by invoking the WSMX Parser. In general, more independent conversations can be running inside WSMX, thus information carried by the context is used to identify the execution semantics associated with the conversation from the context. The execution semantics then passes obtained data to the WSMX Process Mediator.

The role of the WSMX Process Mediator is to decide, which data will be added to which choreography, i.e. requester's or provider's choreography. Please note that choreographies of WSMO services are modeled as Abstract State Machines [1] and are processed using standard algorithms during runtime. Memory of the choreography contains available instance data of ontological concepts. A choreography rule which antecedent matches available data in the memory is selected from the rule base and by execution of the rule's consequent, the memory is modified (data in the memory is updated, deleted or removed). This decision is based on analysis of both choreographies and concepts used by these choreographies and is in detail described in [2]. In our scenario, Process Mediator first updates the memory of the requester's choreography with the information that the Payment Request has been sent. The Process Mediator then evaluates that data should be added to the memory of the provider's choreography.

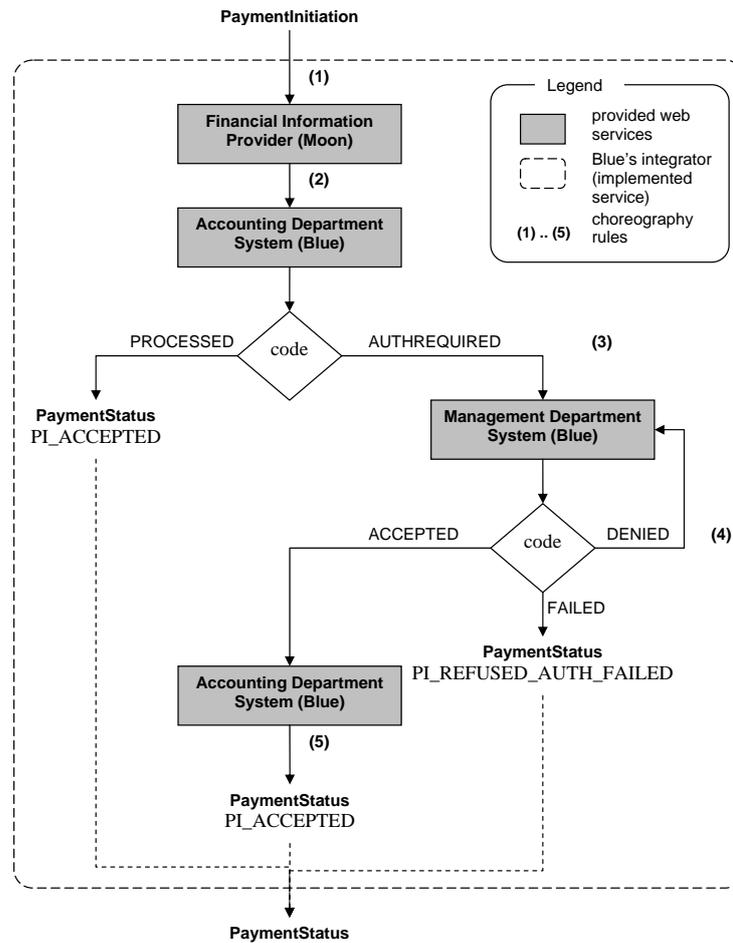
**Choreography Process.** Figure 2 depicts Blue's Payment choreography including rules that are elaborated further on listings provided in this section. First, a controlled instance is initialized during the execution and can be modified only by the choreography execution. Its value attribute belongs to a finite set of states that are used to control the execution. Each rule checks in its condition the controlled instance and is fired only when controlled instance permits.

The rules in Moon Payment choreography specify: a set of variables, a rule conditions for the variable binding and a set of actions operating on the data provided in the variable bindings when rule conditions are satisfied.

The following notation is used in Moon Payment choreography pseudocode: keywords in WSML are marked with bold; "?" followed by an identifier represents a variable; ontology concept names are written in camel case.

**Moon's Financial Information.** During the execution, when the first condition is met (i.e. PaymentInitiation has been sent with the goal), the actions of the rule 1 can be executed. The new BankingInformationRequest instance is an input to the Moon's Financial Information Provider service, the invocation of which results in banking information (Moon's bank account details) response message.

Listing 1.1 shows rule 1 in full WSML syntax. However, for the brevity we present the rest of the rules in more concise, pseudocode form as shown on Listing 1.2.



**Figure 2.** Blue's Payment Scenario Choreography

```

forall {?controlled, ?request} with (
  ?controlled[oasm#value hasValue oasm#InitialState] memberOf oasm#ControlState and
  ?request memberOf pay#PaymentInitiation)
do
  add(_#[moon#hasRequestId hasValue "token.id"] memberOf moon#BankingInformationRequest)
  delete(?controlled[oasm#value hasValue oasm#InitialState])
  add(?controlled[oasm#value hasValue oasm#State1])
endforall
  
```

**Listing 1.1.** Rule Creating BankingInformationReq in WSML Syntax (Rule 1)

```

forall PaymentInitiation ?request do
  create BankingInformationRequest instance
  
```

**Listing 1.2.** Rule Creating BankingInformationReq in Pseudocode (Rule 1)

**Blue’s Accounting Department System Payment Initiation.** The new PaymentInitiationFDRequest instance is an input for the Blue’s Financial Information Department System service and after the invocation the payment initiation response is received. The rule that triggers Blue’s Financial Department System invocation is presented on Listing 1.3.

```
forall PaymentInitiation ?request, BankingInformationResponse ?variable do
  create PaymentInitiationFDRequest instance
```

**Listing 1.3.** Rule Creating PaymentInitiationFDRequest (Rule 2)

If the initial amount is small, the Accounting Department will accept the payment directly and there will be sufficient information to create the final response for the customer and final rule 5 can be fired finishing the choreography execution.

**Blue’s Management Department System First Authorization Request.** If Blue’s Accounting Department requires an authorization code then the authority’s name will have to be determined for the payment approval via Blue’s Management Department System service. It might be required to ask more than once for the authorisation, thus there might be many authorisation responses (some rejecting and some accepting the payment). In order to avoid checking the same data again, the service response instance is flagged as not processed (by setting the attribute to **true**). Also, if an authorisation request is created, the value attribute of the controlled instance is set to identify the authority that was asked to accept the payment. For example, if the payment amount is more than 10.000 Euro and less than 50.000 Euro, the third authority (Arnold Black) will be asked first and the state will be set to "Authorised3".

```
forall PaymentInitiationFDResponse ?variable with response ?code attribute and ?controlled instance do
  if (?code = "PROCESSED") then
    create PaymentStatus instance with code "PL-ACCEPTED"
  if (?code = "AUTHREQUIRED") then
    if (?amount > 1999.99 and ?amount < 3000) then
      create PaymentAuthorizationMDRequest instance with the first authority's name
      add(?controlled[value = "Authorised1"])
      ...
    if (?amount > 49999.99) then
      create PaymentAuthorizationMDRequest instance with the last authority's name
      add(?controlled[value = "Authorised4"])
      add notProcessed attribute to the response of the service with the value "true"
```

**Listing 1.4.** Rule Determining Authority Required for MDS Payment Authorization (Rule 3)

**Blue’s Management Department System Subsequent Authorization Requests.** The rule selects the response instances from the Management Department System service that were not checked yet. If the Management Department System accepts the payment, it will provide an authorisation code, which is used to create a request to the Blue Accounting Department System. Otherwise the request for payment authorisation has to be repeated with higher-rank authority. We can determine which authority was previously asked by looking at the value attribute of the controlled instance. For example, if the

state was "Authorised3", it means that the request to the Management Department System service will be formulated with the forth authority's name and the state will be changed to "Authorised4". Processed response is flagged as "false" (by changing the notProcessed attribute) and (as in rule 3) the latest Management Department System response is marked as not processed. If the previously asked authority was the forth, there is no authority left to ask for payment approval, therefore the final PaymentStatus response is created with the status "PLREFUSED\_AUTH\_FAILED" and the execution ends. The same situation applies if the Management Department System service fails to authorise the request.

```
forall PaymentAuthorizationMDResponse ?response with attributes ?code and notProcessed="true" do
  if (?code = "ACCEPTED") then
    create PaymentInitiationFDRequest instance with the authorisation code of the ?response
  if (?code = "DENIED") then
    add(?response[notProcessed = "false"])
    if (?controlled[value = "Authorised1"]) then
      create PaymentAuthorizationMDRequest instance with the second authority's name
      add notProcessed attribute to the response of the service with the value "true"
      add(?controlled[value = "Authorised2"])
    ...
    if (?controlled[value = "Authorised4"]) then
      create PaymentStatus instance with code "PLREFUSED_AUTH_FAILED"
  if (?code = "FAILED") then
    create PaymentStatus instance with code "PLREFUSED_AUTH_FAILED"
```

**Listing 1.5.** Loop over Blue's MDS Authorization (Rule 4)

**Final Payment Status Response.** This rule will be executed if the authorisation response in rule 4 was "ACCEPTED". There is another PaymentInitiationFDResponse instance in the state ontology as a result of executing rule 1, but its code is "AUTHREQUIRED" (otherwise the execution would have already ended) and it does not have sufficient information to create final response for the Blue client. PaymentInitiationFDResponse instance is selected since it contains header and originalGroupInfoAndStatus attributes in order to create and send the payment status response to the client.

```
forall PaymentInitiationFDResponse ?variable do
  create PaymentStatus instance with code "PLACCEPTED"
```

**Listing 1.6.** Final Payment Status Response (Rule 5)

## 2.2 Purchase Order Mediation Scenario - Data Mediation

Although we chose not to use the functionality of the WSMX Data Mediator in the payment SWS Challenge scenario, we see the recent integration into WSMX as a noteworthy improvement of our system compared to the previous SWS Challenge. SWS Challenge Purchase Order Mediation scenario<sup>8</sup> is more suited to show the added value resulting from descriptive mappings between ontologies

<sup>8</sup> [http://sws-challenge.org/wiki/index.php/Scenario:\\_Purchase\\_Order\\_Mediation](http://sws-challenge.org/wiki/index.php/Scenario:_Purchase_Order_Mediation)  
(previous solution described in [3])

(compared to hard-coded message transformations in adapters). We have successfully used the WSMX Data Mediator for this scenario and thus present the data mediation based on that example in the following paragraphs.

WSMO conceptually takes into account that there may be different implementations for data mediation. WSMO specifies OO-Mediators, which are used as descriptions that define a mediation service between two ontologies (independent of the implementation). An OO-Mediator has an ID, references to source and target ontology, and a mediationService property that points to the service or component that actually implements the data mediation between the given ontologies<sup>9</sup>. Before the choreography execution, the ontologies used in the respective choreographies of goal and web service are inspected. For each pair of goal and web service ontologies ( $GO_i, WO_j$ ) with  $GO_i \neq WO_j$ , it is checked whether there is an OO-Mediator registered. If this is the case, the specified data mediation service is requested using the input data. Any mediation results are then combined and forwarded to choreography execution (if no data mediation was necessary or possible, the unmediated data is used).

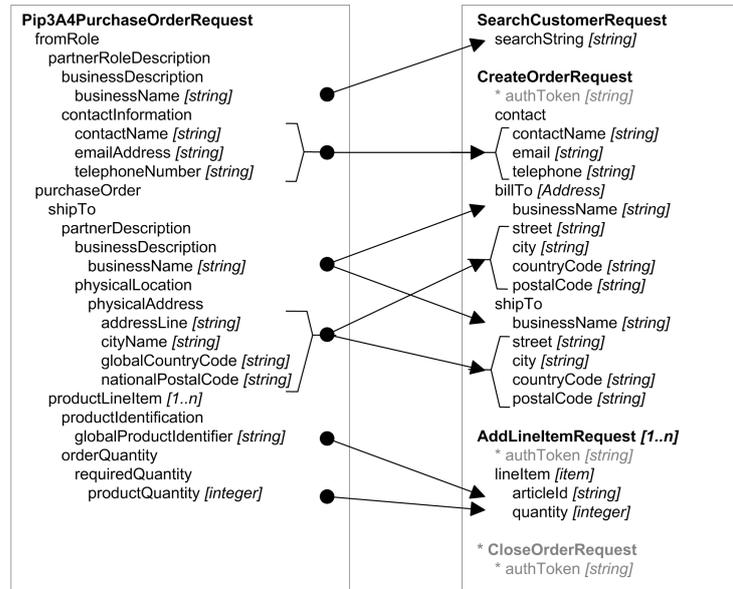


Figure 3. Data Mediation Moon Scenario

To illustrate the example scenario, Figure 3 shows the relevant parts of the RosettaNet expected input instances as well as all of the instances that need to be created during runtime in order to communicate with the Moon legacy

<sup>9</sup> note that OO-Mediators can also be used in other ways, which will be omitted here since it is not relevant to this example

systems. The hierarchy shown depicts instances with their respective attributes, whereas an attribute is member of a concept with the same name except denoted otherwise. The creation of the instances and attributes marked with an asterisk (\*) is not the responsibility of data mediation but of choreography execution, since some of the attributes do not have any correspondence in the input data.

The WSMX Data Mediator itself uses mapping descriptions to implement the data mediation. The mappings between the RosettaNet PIP 3A4 Purchase Order Request and Moon ontologies are created during the *integration setup phase*. They are represented in an abstract, ontology mapping language. The creation of those mappings is a semi-automatic process (due to the requirement of accuracy). The domain expert is aided in this step by a graphical mapping tool<sup>10</sup>. Utilizing different *perspectives* on source and target ontologies allows for the creation of complex mappings using only a simple operation, *map*. A contextualization strategy as well as lexical and structural suggestion algorithms provide further support for the domain expert. The model is formally described and linked to the Abstract Mapping Language (described in [10] and elaborated in [11]). Statements in the Abstract Mapping Language include, amongst others, *classMappings*, *attributeMappings*, *classAttributeMappings* and various conditional statements.

The mappings for this use case can be created using only the *PartOf perspective* of the mapping tool, which focuses on concepts, attributes and attributes' type hierarchies. The generated mapping statements between the RosettaNet BusinessDescription and the Moon SearchCustomerRequest are shown in Listing 1.7 using the Abstract Mapping Language.

```
classMapping(BusinessDescription, SearchCustomerRequest)
attributeMapping([(BusinessDescription) businessName => string], [(SearchCustomerRequest)
    searchString => string])
classMapping(string, string)
```

**Listing 1.7.** Data Mediation Mapping Rules (simplified)

During the *integration runtime phase*, the Abstract Mapping Language statements are converted to WSML rules which specify the conditional creation of instances of the target ontology. The input instances and the rules are registered with a reasoner, along with the source and target ontologies. By querying the knowledge base for instances of the target ontology, the rules fire and thus generate the respective instances of the target ontology. For the given example, the knowledge base is shown in Listing 1.8 (rules and input instances only), the queries and the resulting mediated instances in Listings 1.9 and 1.10. All of these listings are in WSML.

```
axiom m#ccMappingRule4 definedBy
m#mappedConcepts(rosettacore#BusinessDescription, moon#SearchCustomerRequest, ?X3)
and m#mediated1(?X3, moon#SearchCustomerRequest) memberOf moon#SearchCustomerRequest
:-
?X3 memberOf rosettacore#BusinessDescription.
```

<sup>10</sup> Available as part of the Web Service Modeling Toolkit, <http://wsmt.sourceforge.net/>

```

axiom m#aaMappingRule8 definedBy
  m#mediated1(?X5,moon#SearchCustomerRequest)[moon#searchString hasValue ?Y6] memberOf
  moon#SearchCustomerRequest
:-
  ?X5[rosettacore#businessName hasValue ?Y6] memberOf rosettacore#BusinessDescription
  and ?X5 memberOf ?SC7
  and m#mappedConcepts(?SC7,moon#SearchCustomerRequest,?X5).

axiom m#ccMappingRule2 definedBy
  m#mappedConcepts(_string,_string,?X1)
:-
  ?X1 memberOf _string.

instance in#input_BusinessDescriptionFromRolePartDe memberOf rosettacore#BusinessDescription
  rosettacore#businessName hasValue "Blue Company"

```

**Listing 1.8.** Data Mediation Knowledge Base

```

// Query 1:
?x memberOf moon#SearchCustomerRequest
// 1 result:
?x=m#mediated1(in#input_BusinessDescriptionFromRolePartDe,moon#SearchCustomerRequest)

// Query 2 (created based on the result of Query 1):
m#mediated1(in#input_BusinessDescriptionFromRolePartDe, moon#SearchCustomerRequest)[?y
  hasValue ?z] memberOf moon#SearchCustomerRequest and ?z memberOf ?avC.
// 1 result:
?y=moon#searchString, ?z="Blue Company", ?avC="http://www.wsmo.org/wsml/wsml-syntax#
  string"

```

**Listing 1.9.** Data Mediation Queries

```

instance m#mediated1('in#input_BusinessDescriptionFromRolePartDe, moon/SearchCustomerRequest')
  memberOf moon#SearchCustomerRequest
  moon#searchString hasValue "Blue Company"

```

**Listing 1.10.** Data Mediation Final Result

During choreography execution, the additionally required instances and attributes not generated by data mediation are added. Listing 1.11 shows the respective transition rule adding an *authToken* attribute to the Moon SearchCustomerRequest and CreateOrderRequest.

```

forall {?controlstate, ?searchCustReq, ?createOrdReq} with (
  ?controlstate[oasm#value hasValue oasm#InitialState] memberOf oasm#ControlState and
  ?searchCustReq memberOf moon#SearchCustomerRequest and
  ?createOrdReq memberOf moon#CreateOrderRequest
) do
  add(?searchCustReq[moon#authToken hasValue "MaciejZaremba"]) // add authToken
  add(?createOrdReq[moon#authToken hasValue "MaciejZaremba"]) // add authToken
  add(?controlstate[oasm#value hasValue moonc#CreateOrder])
  delete(?controlstate[oasm#value hasValue oasm#InitialState])
endforall

```

**Listing 1.11.** Choreography Example

More information on the WSMX Data Mediator can be found in [6] and [7].

### 3 Conclusion

In this paper we presented our approach to dynamic B2B integration based on the Semantic Web Services technology in particular we have addressed an

extended mediation scenario and provided a generic, rule-based data and process mediation of heterogeneous services. Our solution is a contribution to the SWS Challenge and further, will be part of an evaluation within this initiative.

## Acknowledgments

This work is supported by the Science Foundation Ireland Grant No. SFI/02/CE1/I131, and the EU projects SUPER (FP6-026850), and SemanticGov (FP-027517).

## References

1. E. Börger and R. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag, 2003.
2. E. Cimpian and A. Mocan. WSMX Process Mediation Based on Choreographies. In *Business Process Management Workshops*. 2005.
3. T. Haselwanter, *et al.* Dynamic B2B Integration on the Semantic Web Services. In *Proceedings of the Semantic Web Service Challenge Workshop - Phase II, Workshop at 3rd European Semantic Web Conference (ESWC)*. 2006.
4. T. Haselwanter, *et al.* WSMX: A Semantic Service Oriented Middleware for B2B Integration. In *Proceedings of the 4th International Conference on Service Oriented Computing (ICSOC), Springer-Verlag LNCS*. 2006.
5. M. Kerrigan, A. Mocan, M. Tanler, and D. Fensel. The Web Service Modeling Toolkit - An Integrated Development Environment for Semantic Web Services (System Description). In *Proceedings of the 4th European Semantic Web Conference (ESWC)*. 2007.
6. A. Mocan and E. Cimpian. An ontology-based data mediation framework for semantic environments. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 3(2):66–95, April-June 2007.
7. A. Mocan, M. Kerrigan, and E. Cimpian. Applying reasoning to instance transformation. In *Proceedings of the International Workshop on Ontologies: Reasoning and Modularity (WORM-08)*. June 2008. To appear.
8. 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.
9. D. Roman, *et al.* Web Service Modeling Ontology. *Applied Ontologies*, 1(1):77 – 106, 2005.
10. F. Scharffe and J. de Bruijn. A language to specify mappings between ontologies. In *Proceedings of the Internet Based Systems IEEE Conference (SITIS05)*. 2005.
11. F. Scharffe, J. Euzenat, and L. Serafini. Specification of delivery alignment format. Knowledge web deliverable d2.2.6, Knowledge Web Consortium, 2006.
12. T. Vitvar, *et al.* Semantically-enabled service oriented architecture: Concepts, technology and application. *Service Oriented Computing and Applications, Springer London*, 1(2), 2007.