DIP
Data, Information and Process Integration with Semantic Web Services

FP6 – 507483

Deliverable

WP5: Service Mediation
D5.8b
Final Prototype of Mediation and Composition in a Real World Scenario

Christian Drumm and Andreas Friesen and Jens Lemcke and Daniel Oberle

November 24, 2006
SUMMARY

D5.8 discusses the need for combining Web service mediation and composition techniques in a real world scenario. The real world scenario we apply here is the frequently occurring carrier shipper scenario. So far, solutions for the carrier shipper interaction have been of a rather static nature. The different parties involved face several difficulties that require a great deal of manual development. The envisioned Enterprise Services Architecture (ESA) of SAP proposes to remedy such problems by enabling a service-oriented way of business partner interaction. However, we argue that several challenges remain for the developer despite the advantages of ESA. In our scenario, the challenges are i) the manual creation of a collaborative business process between shipper and carrier, ii) inflexibility wrt to prescribed carrier interfaces, and iii) no means for run time selection of a carrier. In this deliverable we focus on i) and ii) and discuss the rationale of our prototype, an enhanced process modelling tool, to support the automatic generation of collaborative business processes which boils down to an automatic composition of two public processes of a shipper and a carrier. In turn, the automatic composition requires different types of mediation. We contribute a lifting engine and mapping engine to solve the mediation problems. The novel approaches rely on a domain ontology to improve lifting and mapping results. We propose a mediation from existing WSDL files to the ontological level such that the ILOG composer can be reused in order to achieve the automatic composition. Furthermore, we reuse an existing SAP tool for graphical process modelling as well as a SAP process repository.

With our solution, we show how results of DIP can be beneficially applied in a real-world scenario. Therefore, our tool makes use of the “Web Services Modeling Ontology” (WSMO, W3C member submission), its open source Semantic Web Services Architecture and the implementation of its components on top of an existing, real-life business process management (BPM) application. The tool thus exploits DIP technology into a current corporate environment. It is expected that our implementation should act as a demonstrator for applying DIP results in real-world scenarios. The knowledge gained when integrating advanced semantic with existing syntactic infrastructures will be helpful for further exploitation activities of DIP results. Due to its exemplary intention, this deliverable should be read by persons willing to apply DIP results in further real-world integration scenarios.

The work presented in this deliverable mainly bases on the work performed in the two workpackages WP5 (“Service Mediation”) and WP4 (“Service Usage”). In particular, this deliverable combines the results from D4.12 (“Goal-oriented SWS composition module specification”) with results from D5.1 (“Report on the state-of-the-art and requirements analysis”) and D5.2 (“Business data level mediation module specification”). The need for the combination of these both technologies becomes necessary in this special context of deploying advanced semantic features in traditional syntactic Web service infrastructures. The content of this deliverable is designed in a way that it will be easily integrable with the work done in D5.7a (“First Prototype of Mediation and Discovery in a Real World Scenario”).

Disclaimer: The DIP Consortium is proprietary. There is no warranty for the accuracy
or completeness of the information, text, graphics, links or other items contained within this material. This document represents the common view of the consortium and does not necessarily reflect the view of the individual partners.
## Document Information

<table>
<thead>
<tr>
<th>IST Project Number</th>
<th>FP6 – 507483</th>
<th>Acronym</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Title</td>
<td>Data, Information, and Process Integration with Semantic Web Services</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project URL</td>
<td><a href="http://dip.semanticweb.org/">http://dip.semanticweb.org/</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Document URL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EU Project Officer</td>
<td>Brian Macklin</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Deliverable Number**: 5.8b  
**Title**: Final Prototype of Mediation and Composition in a Real World Scenario  
**Work Package Number**: 5  
**Title**: Service Mediation

<table>
<thead>
<tr>
<th>Date of Delivery</th>
<th>Contractual</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>version 1.0</td>
<td>final</td>
</tr>
<tr>
<td>Nature</td>
<td>prototype ☑ report ☐ dissemination ☐</td>
<td></td>
</tr>
<tr>
<td>Dissemination Level</td>
<td>public ☑ consortium ☐</td>
<td></td>
</tr>
</tbody>
</table>

**Authors (Partner)**: Stephan Grimm (FZI), Kioumars Namiri (SAP)  
**Resp. Author**: Daniel Oberle  
**E-mail**: d.oberle@sap.de  
**Partner**: SAP  
**Phone**: +49 (0) 721 6902-85

**Abstract (for dissemination)**: Please fill in the dissemination abstract summary using the ‘abstract’ command.  
**Keywords**: business data, ontology, ontology building methodology, ontology design principles

### Version Log

<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>05-10-06</td>
<td>1</td>
<td>Daniel Oberle</td>
<td>Initial version</td>
</tr>
<tr>
<td>11-24-06</td>
<td>2</td>
<td>Daniel Oberle</td>
<td>Final version</td>
</tr>
</tbody>
</table>
### Project Consortium Information

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| National University of Galway                | NUIG    | Prof. Dr. Christoph Bussler  
Digital Enterprise Research Institute (DERI)  
National University of Ireland, Galway  
Galway  
Ireland  
E-mail: chris.bussler@deri.ie  
Tel: +353 91 512460 |
| Fundacion De La Innovacion. Bankinter        | Bankinter | Monica Martinez Montes  
Fundacion de la Innovation. BankInter  
Paseo Castellana, 29  
28046 Madrid,  
Spain  
Email: mmmez@bankinter.es  
Tel: 916234238 |
| Berlecon Research GmbH                       | Berlecon | Dr. Thorsten Wichmann  
Berlecon Research GmbH,  
Oranienburger Str. 32,  
10117 Berlin, Germany  
E-mail: tw@berlecon.de  
Tel: +49 30 2852960 |
| British Telecommunications Plc.              | BT      | Dr. John Davies  
BT Exact (Orion Floor 5 pp12)  
Adastral Park Martlesham  
Ipswich IP5 3RE,  
United Kingdom  
Email: john.nj.davies@bt.com  
Tel: +44 1473 609583 |
| Swiss Federal Institute of Technology, Lausanne | EPFL    | Prof. Karl Aberer  
Distributed Information Systems Laboratory  
École Polytechnique Fédérale de Lausanne  
Bât. PSE-A  
1015 Lausanne, Switzerland  
E-mail: Karl.Aberer@epfl.ch  
Tel: +41 21 693 4679 |
| Essex County Council                         | Essex   | Mary Rowlett  
Essex County Council,  
PO Box 11, County Hall, Duke Street,  
Chelmsford, Essex, CM1 1LX,  
United Kingdom.  
E-mail: maryr@essexcc.gov.uk  
Tel: +44 (0)1245 436524 |
| Forschungszentrum Informatik                 | FZI     | Andreas Abecker  
Forschungszentrum Informatik  
Haid-und-Neu Strasse 10-14  
76131 Karlsruhe  
Germany  
E-mail: abecker@fzi.de  
Tel: +49 721 96540 |
<table>
<thead>
<tr>
<th>Company</th>
<th>Contact Information</th>
</tr>
</thead>
</table>
| Institut für Informatik, Leopold-Franzens Universität Innsbruck (UIBK) | Prof. Dieter Fensel  
Institute of computer science  
University of Innsbruck  
Technikerstr. 25  
A-6020 Innsbruck, Austria  
Email: dieter.fensel@deri.org  
Tel: +43 512 5076485 |
| ILOG SA                                      | Christian de Sainte Marie  
9 Rue de Verdun, 94253, Gentilly, France  
E-mail: csma@ilog.fr  
Tel: +33 1 49082981 |
| inubit AG                                    | Torsten Schmale, inubit AG, Lützowstraße 105-106  
D-10785 Berlin, Germany  
E-mail: ts@inubit.com  
Tel: +49 30726112 0 |
| Intelligent Software Components, S.A. (iSOCO) | Dr. V. Richard Benjamins, Director R&D  
Intelligent Software Components, S.A.  
Pedro de Valdivia 10  
28006 Madrid, Spain  
E-mail: rbenjamins@isoco.com  
Tel: +34 913 349 797 |
| Net Dynamics Internet Technologies GmbH u. Co KG | Peter Smolle  
Net Dynamics Internet Technologies GmbH u. Co KG  
Prinz-Eugen-Strasse 68-70  
A-1040 Wien, Austria  
E-mail: peter.smolle@netdynamissterch.com  
Tel.: +43 1 503982615 |
| The Open University                          | Dr. John Domingue  
Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK  
E-mail: j.b.domingue@open.ac.uk  
Tel.: +44 1908 655014 |
| SAP AG                                       | Dr. Elmar Dorner  
SAP Research, CEC Karlsruhe  
SAP AG  
Vincenz-Priessnitz-Str. 1  
76131 Karlsruhe, Germany  
E-mail: elmar.dorner@sap.com  
Tel: +49 721 6902 31 |
| Sirma AI Ltd.                                | Atanas Kiryakov, Ontotext Lab, - Sirma AI EAD, Office Express IT Centre, 3rd Floor  
135 Tzarigradsko Chausse, Sofia 1784, Bulgaria  
E-mail: atanas.kiryakov@sirma.bg  
Tel.: +359 2 9768 303 |
| Tiscali Österreich GmbH | Dr Dieter Haacker  
Tiscali Österreich GmbH.  
Diefenbachgasse 35,  
A-1150 Vienna, Austria  
E-mail: Dieter.Haacker@at.tiscali.com  
Tel: +43 1 899 33 160 |
|-----------------------|------------------------------------------------------------------|
| Unicorn Solution Ltd. | Jeff Eisenberg  
Unicorn Solutions Ltd,  
Malcha Technology Park 1  
Jerusalem 96951,  
Israel  
E-mail: Jeff.Eisenberg@unicorn.com  
Tel.: +972 2 6491111 |
| Vrije Universiteit Brussel | Carlo Wouters,  
Starlab- VUB  
Vrije Universiteit Brussel  
Pleinlaan 2, G-10  
1050 Brussel, Belgium  
E-mail: carlo.wouters@vub.ac.be  
Tel.: +32 (0) 2 629 3719 |
# Table of Contents

1 **Introduction**  

2 **Carrier Shipper Scenario**  
   2.1 As-Is Situation  
      2.1.1 Description  
      2.1.2 Pain Points  
   2.2 SAP Enterprise Services Architecture  
      2.2.1 Description  
      2.2.2 The Carrier Shipper Scenario in ESA  
      2.2.3 Challenges  

3 **Prototype**  
   3.1 Design  
      3.1.1 Preliminaries  
      3.1.2 Architecture  
      3.1.3 Mediation  
      3.1.4 Composition  
   3.2 Implementation  
      3.2.1 Lifting Engine  
      3.2.2 Mapping Engine  
      3.2.3 Domain Ontology  
      3.2.4 Composition Engine  
      3.2.5 Graphical User Interface  
      3.2.6 Process Repository  

4 **Conclusion**
1 Introduction

This deliverable discusses the need for combining Web service mediation and composition techniques in a real world scenario. The real world scenario we apply here is the frequently occurring carrier shipper scenario. It is part of the standard SAP order-to-cash process in the logistics domain. A customer typically places a sales order with a shipper, such as amazon.com, which delegates item delivery to carriers such as UPS or DHL.

So far, solutions for the carrier shipper interaction have been of a rather static nature. The different parties involved face several difficulties. For example, the publication of new carrier services is difficult, because the process with an individual carrier is firmly implemented in the specific SAP system configuration of the shipper. A new service requires to enter this implementation. In essence, a great deal of manual development is necessary.

The envisioned Enterprise Services Architecture (ESA) of SAP proposes to remedy such problems by enabling a service-oriented way of business partner interaction. The goals are to enable flexibility in markets, high adaptability of internal company IT structures, simplification of interoperation with business partners as well as support for integration and outsourcing of business units. However, we argue that several challenges remain for the developer despite the advantages of ESA. In our scenario, the challenges are i) the manual creation of a collaborative business process between shipper and carrier, ii) inflexibility wrt to prescribed carrier interfaces, and iii) no means for run time selection of a carrier.

In this deliverable we focus on i) and ii) and discuss the rationale of our prototype, i.e., an enhanced process modelling tool. The prototype supports automatic generation of collaborative business processes which boils down to an automatic composition of two public processes of a shipper and a carrier. The automatic composition requires different types of mediation: the lifting of the public processes of the two partners which in turn requires the lifting of the input and output messages associated with the process steps, and the lifting of the process descriptions itself, as well as the automatic generation of message mappings.

We contribute a lifting engine and mapping engine to solve the mediation problems. The novel approaches rely on a domain ontology to improve lifting and mapping results. We propose a mediation from existing WSDL files to the ontological level such that the ILOG composer [1] can be reused in order to achieve the automatic composition. Furthermore, we reuse an existing SAP tool for graphical process modelling as well as a SAP process repository [6].

The document is structured as follows: Chapter 2 discusses the as-is situation of how the carrier shipper scenario is typically realized as well as ESA and its challenges. Subsequently, chapter 3 presents the design and implementation of our enhanced process modelling tool. We conclude and provide an outlook in chapter 4.
2 CARRIER SHIPPER SCENARIO

The "real world scenario" we discuss in this deliverable is the frequently occurring carrier shipper scenario. It is part of the standard SAP order-to-cash process in the logistics domain. The scenario, depicted in Fig. 2.1, involves three parties: i) a customer, ii) a shipper, and iii) a carrier. As illustrated, the customer first places a sales order with the shipper, which enters it in its local SAP system. After that, the appropriate steps for delivery and picking and packing are taken. After sending the shipping information to the Express Shipping Interface (XSI), the goods are labeled and the manifest is sent to the carrier. This is the trigger for the actual shipping by the carrier which is tracked by the shipping process step in the SAP system of the shipper. The last activity is the execution of the billing process. Sometimes, carriers change their conditions of service. In this case, they need to notify the shipper of those updates. Also, for a new contractual cooperation between a shipper and a carrier, the new conditions need to be input into the shipper's system.

In the following, we discuss the as-is situation of how this scenario is typically realized. Thus, section 2.1 describes the current solution and identifies its pain points. Most of the pain points will be addressed by SAP’s Enterprise Services Architecture (ESA). Section 2.2 introduces ESA and argues that, although ESA remedies most problems, there still remain challenges for developers. These challenges are the motivation for our work.

2.1 As-Is Situation

This section discusses the current solution of how the carrier shipper interaction is typically realized. The rather static nature of the solutions leads to several pain points which are identified afterwards.

2.1.1 Description

In the following, we assume that the functionality of the carriers is provided via Web service interfaces and that the shipper uses a SAP solution. The SAP solution of the shipper is used to integrate the systems of the different parties. However, as connotated in Fig. 2.1, not all process steps are directly used as they are pre-existent in the SAP solution.

Some process steps need configuration in order to fit the needs of the respective SAP customer (here: the shipper) for its specific business case. This is illustrated by the "customer enhancements" in Fig. 2.1. Also, the process steps of labeling, preparing the manifest, and the shipping itself are influenced by the specific requirements of the respective carrier. Therefore, these steps are mainly implemented by the carriers.

Typically SAP solutions provide the main business functionality because it is similar in most companies. Specific adoption to special requirements need to be done by extending the current SAP product via its interfaces in a static manner.

2.1.2 Pain Points

Due to the static nature of current solutions, the different parties involved face several difficulties. These are explained in the following.
Carriers

On the carrier side the following problems occur:

**Difficulty to Publish New Services** The publication of new services is difficult, because the process with an individual carrier is firmly implemented in the specific SAP system configuration of the shipper. A new service requires to enter this implementation therefore manual development is necessary.

**Difficulty to Notify Customers of Service Update** For this action, the carrier needs to interact with the shipper. If the change of service implies the adoption of the process between both partners, they face the same problem as in the former case.

**Difficulty to Connect to SAP Systems** In order to connect a third party system to a SAP system in general message mappings are necessary. In addition custom development might be necessary to adapt the business process of the carrier to fit the frame the SAP system sets.

**Shippers**

**Difficulty to Add or Update Services** Due to the firm implementation of the processes, this problem also appears to the shipper.

**Difficulty to Change Service Provider** Since processes are firmly coded into the SAP system, the change from an existing carrier with an already implemented process to a new business partner is difficult. The cost for changing carriers are quite high due to the implementation effort to take.
Different Service Interfaces Provided by Carriers  The setup of a process with different carriers requires always new effort, because little knowledge can be reused due to their different interfaces.

Difficulty to Compare Services Provided by Carriers  The diversity of interfaces and properties (like pricing model) of the services provided by the carriers makes their comparison difficult.

2.2 SAP Enterprise Services Architecture

The envisioned Enterprise Services Architecture of SAP proposes a service-oriented way of business partner interaction. The goals are to enable flexibility in markets, high adaptability of internal company IT structures, simplification of interoperation with business partners as well as support for integration and outsourcing of business units. In this section, we first introduce the reader to ESA and show how the carrier shipper interaction will be solved in ESA. We conclude that most of the pain points identified in the previous section will be erased. However, several challenges remain for the developer which we address by our prototype.

2.2.1 Description

SAP is a provider of business software supporting enterprises to perform their more or less standardized business tasks as efficient as possible. The main target is to lower the Total Cost of Ownership (TCO) of their SAP business solutions required to carry out the business activities. The main challenge with respect to business tasks in today’s companies is to keep track of increasingly dynamic markets. The changing business environment demands for more flexibility of the companies to adapt to changing market requirements. Therefore, SAP provides customers with solutions meeting this demand for flexibly performing business.

The demand for flexibility in markets results in the need for high adaptability of internal company IT structures, simplification of interoperation with business partners as well as support for integration and outsourcing of business units, respectively. At the same time, prior investments of SAP customers in their existing IT infrastructure should be leveraged.

In order to provide support for the market requirements mentioned above, SAP’s vision is to build an integrated platform for individual service providers and requesters. The facilitating SAP technology is called Business Process Platform. It provides a uniform means for companies to represent the interfaces of their business functionality by Enterprise Services which are fully based on the Web Service Description Language (WSDL) standard. Enterprise Service interfaces can be built on top of existing systems encapsulating their functionality, thus leveraging prior investments. Furthermore, existing Enterprise Services can be re-used to build more complex Enterprise Service and thus support flexibility of internal company structures and ease of integration and outsourcing with other partners using the SAP xApps technology.

The facilitator of this Business Process Platform is the design of the SAP products in the Enterprise Services Architecture (see Fig. 2.2). It defines a clear structuring of the SAP software architecture into separate layers. SAP NetWeaver basically acts as application server and technical integration platform for different business partners.
The mySAP Business Suite provides the SAP business software in an Enterprise-Service-enabled way while xApps technology allows for simplified composition of new functionality re-using existing applications or parts of those. The Business Solutions layer provides a clear documentation on the different components of the SAP solutions from a manager’s perspective.

2.2.2 The Carrier Shipper Scenario in ESA

In this section we provide a description of how the carrier shipper interaction would be realized with the Enterprise Services Architecture. An ESA-based implementation would involve three main components: i) A process modeling tool, ii) a process repository, and iii) a process execution engine capable of executing the modeled processes (cf. Figure 2.3).

Focusing on the shipper we now describe the steps necessary to create an executable process in ESA. Using the process modelling tool, a domain expert would create a graphical representation of the process executed at the shipper whenever a new sales order is processed. Note that this graphical representation is at first not linked to any of the shippers’ business systems or any carrier services. Therefore in a second step, the domain expert manually connects the single process steps of the business process to services offered by either the internal or the partner business systems. Connecting different services and systems usually requires a mapping between different message formats. Consequently the domain expert also needs to create the necessary mappings converting between the input and output messages of the different business systems involved.

Services available in the business systems of partners usually need to be invoked in a certain order as they are involved in the internal processes of the partner. In our scenario, for example, it does not make sense to invoke a “track and trace” service of a carrier before a shipping request has been sent to that carrier. However, partners
interacting during a business process only want to make parts of their internal processes visible to the outside. Therefore the notion of private and public processes becomes necessary. A private process is the detailed, internal process of a partner. Based on this process, a view — the public process — can be created. This public process hides all the confidential process details and only shows the process steps that are required for an interaction. The public process is then used by business partners to create a Collaborative Business Process (CPB) involving both internal and external business services (see Fig. 2.4).

After the business process has been manually modeled and the involved business systems have been connected to the different process steps, the business process is stored into the process repository. During run-time a process execution engine retrieves the process from the repository and executes an instance of the process for each incoming sales order.

The main advantage here is the design-time flexibility. After the business process has been modeled using the graphical editor, services implementing different process steps can easily be exchanged during design-time. Integrating a new carrier into the systems for example only requires the connections of the carrier services to the appropriate process steps as well as the development of the necessary message transformations. Concluding, many pain points of current solutions (as discussed in 2.1) can be remedied by the Enterprise Services Architecture. However, there also remain challenges for the developer which we identify in the subsequent section.

2.2.3 Challenges

The attentive reader is already aware of the challenges that remain despite the design-time flexibility of ESA. Below we list three challenges, viz., the still manually executed
and error-prone tasks of composition, discovery, and mediation. We counter the three challenges by applying semantic technologies, i.e., ontological modelling and reasoning, to achieve even greater flexibility.

**Composition** Composing the public process of partners to a *Collaborative Business Process* (CBP) has to be done manually. This is a tedious task given that public processes can become complex. In an ideal case a CBP would be automatically suggested to the developer.

**Discovery** With ESA we do not have flexibility during run-time with respect to the dynamic exchange of carriers, because the process execution engine runs predefined processes only. Our goal is to enable run-time discovery of carriers based on the availability of their services.

**Mediation** Interaction between enterprise services usually requires a mediation between different message formats. We can distinguish between mediation of inputs and outputs as well as lifting existing XML data to an ontological level.

**Inputs and Outputs** So far, mediation between different message formats required a domain expert who needs to create the necessary mappings between the input and output messages of the different business systems involved. This is a very time-consuming and error-prone task. Automatic composition of the CBP will require to dynamically mediate between inputs and outputs allowing arbitrary carrier interfaces.

**Lifting** Existing XML-based information has to be "lifted" to the ontological level when countering the challenges above with semantic technologies. We can distinguish two types of lifting:
**Process Descriptions** For automatically suggesting a CBP available WSDL descriptions of the carrier services need to be transformed to an ontological format. The composer technology we are going to use is based on the Semantic Web Services composition approach.

**XML Messages** For automatic discovery of carrier services at run-time, XML messages of the shipper need to be transformed to an ontological format. We will use technology which is based on the Semantic Web Services automatic discovery approach.

In this deliverable we show how the automatic composition (i.e., generation of the CBP) in the carrier shipper scenario can be achieved. Accordingly, we address the corresponding mediation problems (inputs and outputs as well as lifting of process descriptions). In contrast, deliverable 5.7 focusses automatic discovery of carriers at run time and the corresponding mediation of XML messages.
3 Prototype

This chapter presents our prototype of an enhanced process modelling tool. It addresses the challenge of automatically generating a CBP including several kinds of mediation problems that occur. We discuss the design in section 3.1 and subsequently some interesting implementation issues in 3.2.

3.1 Design

Using technologies developed in the semantic Web services area allows us to challenge of automatically generating a CBP along all the associated mediation problems. We start with some preliminary definitions in 3.1.1 and continue by describing the architecture for our enhanced process modelling tool (section 3.1.2). Following this, we detail the solution to the mediation problems (section 3.1.3) as well as the automatic generation of the CBP (section 3.1.4).

3.1.1 Preliminaries

Before we start with the description of our solution we first need to define some elementary terms which we will use during the later discussions.

Definition 3.1.1 (Ontology) An ontology $O$ is a structure

$$O := (C, R, A, T, \leq_C, \leq_R, \leq_A, \leq_T, \sigma_R, \sigma_A)$$

where:

- $C$ is a set of concepts aligned in a hierarchy $\leq_C$
- $R$ is a set of relations aligned according to $\leq_R$
- $A$ is a set of attributes aligned according to $\leq_A$
- $T$ is a set of data types aligned according to $\leq_T$
- $\sigma_R: R \rightarrow C \times C$ is the signature of $R$
- $\sigma_A: A \rightarrow C \times T$ is the signature of $A$.

In addition to that we define the domain of $r \in R$ as $\text{dom}(r) := \pi_1(\sigma_R(r))$ and the range as $\text{range}(r) := \pi_2(\sigma_R(r))$. This definition of ontologies is based on [9].

Definition 3.1.2 (XML Schema) An XML schema is a structure

$$S := (E, A, CT, ST, \delta_c, \alpha_c, \delta_a, \Gamma)$$

where:

- $E$ is a set of element names
- $A$ is a set of attribute names
• CT is a set of complex type names
• ST is a set of simple type names
• a function $\delta_e : E \rightarrow (CT \cup ST)$ defining the data type of an element $e \in E$
• a function $\delta_a : A \rightarrow ST$ defining the data type of an attribute $a \in A$
• a function $\alpha_e : E \rightarrow \mathcal{P}(A)$ defining the set of attributes for each element $e \in E$
• a regular tree grammar $\Gamma$ specifying the structure of a valid XML schema

Definition 3.1.3 (Mapping) A mapping $map$ between two structures $S$ and $T$ is defined as set of mapping elements $me$. Each mapping element relates entities of the source structure $e_{S,i}, \ldots, e_{S,j}$ to entities of the target structure $e_{T,m}, \ldots, e_{T,n}$ using a mapping expression. The mapping expression specifies how entities of the source structure and the target structure are related.

$$\text{map}_{S \rightarrow T} = \{me\}$$
$$me = (e_{S,i}, \ldots, e_{S,j}, e_{T,m}, \ldots, e_{T,n}, \text{mapexp})$$

This definition of mapping has a number of important implications. First a mapping is unidirectional as indicated by the arrow form $S$ to $T$ in our definition and cannot easily be inverted. Second the definition of a mapping given above does not restrict the relation between entities of the source and the target structure to be 1 : 1, but rather allows for $m : n$ relations. Third the nature of the involved mapping expressions is not further restricted. This is due to the fact, that it depends strongly on the type of the source and target structure involved.

Definition 3.1.4 (Alignment) An alignment $A_{S \rightarrow O}$ from an XML schema $S$ to an ontology $O$ is a mapping with:

$$A_{S \rightarrow O} = \{(e_{S,i}, \ldots, e_{S,j}, e_{O,m}, \ldots, e_{O,n}, \text{mapexp})\}$$

$$e_{S,x} \in E_S \cup A_S$$
$$e_{O,y} \in C_O \cup R_O \cup A_O$$

3.1.2 Architecture

After loading the public processes of a carrier and a shipper, the enhanced process modelling tool generates the CBP automatically (if possible) and presents it to the user. Furthermore, the tool generates the message mappings necessary for invoking the involved Web service. Figure 3.1 shows the architecture of our enhanced process modelling tool. We assume, that the representations of the two public processes not only contain the process flow but also the XSDs of the in- and output messages associated with each process step. After loading the two public processes specifications into our tool, the lifting engine generates i) an alignment between the message elements of the XSDs and the domain ontology, and ii) a semantic description of the public processes. In the next step the mapping engine uses the alignments between the ontology and the XSDs to generate a list of possible mappings. Now the composition engine takes the list of possible mappings and the semantic process descriptions to generate
the CPB which is finally presented to the user via a graphical user interface. After a check by the user and possible modifications of the CBP the result is stored into the central process repository. Details on how the lifting, the mapping generation and the composition are performed will be given in subsequent sections.

Figure 3.1: The architecture of the enhanced process modelling tool consists of a lifting engine, a mapping engine, a composition engine, a graphical user interface (GUI), and a process repository (not shown in picture). A domain ontology is used to improve lifting, mapping, and composition.

3.1.3 Mediation

The design of our process modelling tool requires three types of mediation: the lifting of the public processes of the two partners which in turn requires i) the lifting of the input and output messages associated with the process steps, and ii) the lifting of the process descriptions itself, as well as iii) the automatic generation of message mappings. All three types are discussed below.

Lifting Input and Output Messages

In order to create an alignment between the domain ontology and the in- and output messages the lifting component executes a set of elementary matching algorithms. These matching algorithms exploit the information available in the XML schema and the ontology (e.g., element and concept names) to create a similarity matrix. This similarity matrix associates each pair of XML schema and ontology entities \((e_S, e_O)\) with \(e_S \in E \cup A\) and \(e_O \in C \cup R \cup A\) with a similarity value. Based on this similarity matrix an alignment between the XML schema and the domain ontology can be calculated.

Creating Semantic Process Descriptions

For the automatic process composition by the composition engine connotated in Fig. 3.1, the public process description of the shipper as well as the available WSDL descrip-
tions of the carrier services need to be transformed to a format that the composer can work with. This step will be detailed in this section.

**Format**  The composer technology we are going to use bases on the semantic Web services composition approach described in [2] and [3]. This approach will be detailed later on. For now, it is enough to know that for each partner which is to be integrated in the composed process, we need a semantic Web service interface description consisting of the following parts:

- The messages communicated by the semantic Web service given as ontology concepts, and
- Behavioral constraints between the single message exchanges of the semantic Web service.

In other words, the behavioral constraints can be understood as a workflow diagram, like an UML 2.0 Activity Diagram, containing control nodes, like decision, merge, fork and join. The activities in this diagram would be connected to input and output nodes representing the messages communicated. Here, each message is not understood as a technical XML schema description, but an ontology concept for that later on the corresponding XML schema can be nominated.

**Shipper**  The information that is available for the shipper partner is its public process and the links of the public process steps to specific WSDL operations. From this information, we build the semantic Web service description as follows:

1. Each WSDL operation becomes represented by an input node and an output node connected via a sequential control edge. We refer to this construct as a semantic Web service operation.

2. The data communicated by the input and output node are represented by ontology concepts that are obtained by the alignment step of the Lifting engine from the corresponding WSDL operation’s XML schema (cmp. Fig. 3.1).

3. The semantic Web service operations are connected by semantic Web service behavioral constraints that resemble the workflow of the public process of the shipper.

**Carrier**  For the carrier services, the only information we can get are the WSDL files of their Web services. From the lifting component, we again get the alignment of XML schema types communicated to ontology concepts. What is missing however, is a representation of causal interdependencies between the operations that we can use to create the semantic Web service behavioral constraints. Therefore, we build a trivial workflow that is required as input to the composer component. Details follow.

1. Each WSDL operation becomes represented by an input node and an output node connected via a sequential control edge. We refer to this construct as a semantic Web service operation.

2. The data communicated by the input and output node are represented by ontology concepts that are obtained by the alignment step of the Lifting engine from the corresponding WSDL operation’s XML schema (cmp. Fig. 3.1).
3. The semantic Web service behavioral constraints describe a workflow that consists of a fork and a join node. Each branch of this fork-join construct contains exactly one semantic Web service operation.

**Automatic Generation of Message Mappings**

The automatic generation of message mappings is performed by the mapping engine. This component takes the alignments created by the lifting engine as input and generates executable mappings between XML schemas. In order to create a mapping between $S_1$ and $S_2$, the mapping engine takes the alignments $A_{S_1 \rightarrow O}$ and $A_{S_2 \rightarrow O}$ as input. For each mapping element in $A_{S_1 \rightarrow O}$ the mapping engine searches for a mapping element in $A_{S_2 \rightarrow O}$ that relates a schema entity of $S_2$ to the equivalent ontology entity. If such an entity is found, the mapping expression is used to determine how the schema entities of $S_1$ and $S_2$ are related. This in turn creates a new mapping expression that is added to the mapping $map_{S_1 \rightarrow S_2}$.

Note that mappings are not generated between each pair of schemas but only between input schemas of one public process and output schemas of the other and vice versa.

### 3.1.4 Composition

As connotated in Fig. 3.1, the automated process composition by the composition engine is the final step before presenting the suggested CBP to the user via the graphical user interface.

**Task**

For creating the CBP, we use the technology described in [2] and [3]. The main observation behind this technology is that business partners follow their own business processes. These business processes consist of several process steps that on the one hand exchange data amongst themselves, and on the other hand also collaborate with their partners. This however means, that, in order to integrate two business partners with each other, their business processes need to be interconnected. Here, it is important to understand that a business process cannot be handled like an atomic transaction. In a process step, the company can have interactions with its partner before proceeding to the next step, which again results in some communication before it enters the next process step, and so forth.

The integration of two business partners in a message-based communication environment, such as SAP’s Enterprise Services Architecture, can however only work on the basis of atomic transactions. As an additional requirement, this integration must follow the sequential constraints of the message exchanges defined by both parties’ business processes. We call these sequential constraints “behavioral constraints” or just the “behavior” of a business partner’s systems. Please note, that such information cannot be represented by traditional Web service descriptions using WSDL. For an automatic creation of such an integration (the CBP), a partner’s behavior needs to be explicitly specified in a machine-processable manner. This is why the observable part of a business partner’s behavior appears in semantic Web service descriptions. The main task of a composer after all is to combine these sets of behavioral constraints to a combined business process of all parties involved.
Realization

The inputs for the composer component in general are the semantic Web service descriptions of the participating business partners. In Sect. 3.1.3, we explained how to create these descriptions for the two parties shipper and carrier. The composition engine basically compares the inputs and outputs that are defined as ontology concepts in the two behavior descriptions and connects them where possible. For the decision whether a connection is possible, the composer relies on the results from the preceding alignment step. The alignment works in a way that it connects those XML schema elements that it later can generate a mapping for to the same ontology element. Therefore, the composer just needs to look for equivalent concepts in the two behavior descriptions that it can connect.

In the following, we define how to check for the connectability of pairs of outputs and inputs. An output and an input can stem from a process step of the same or different business partners. We call the party owning the output message “sender” and the one owning the input message the “receiver”. For the respective XML schemas, we write $S_{\text{out}}$ and $S_{\text{in}}$.

For the evaluation of the suitability of an output and an input concept, the composer has to consider two sets of alignments: The one of the sender $A_{S_{\text{out}} \rightarrow O}$, and the one of the receiver $A_{S_{\text{in}} \rightarrow O}$. An output concept can only be connected to an input concept, if the whole XML schema corresponding to the input concept ($S_{\text{in}}$) is part of the XML schema corresponding to the output concept ($S_{\text{out}}$). For the condition, whether an XML schema $S_{\text{in}}$ is part of an XML schema $S_{\text{out}}$, we write $\text{partOf}(S_{\text{in}}, S_{\text{out}})$. We are now prepared to formally define the suitability of two concepts.

**Definition 3.1.5 (Suitability)** An input concept $c_{\text{in}}$ suits an output concept $c_{\text{out}}$, iff the receiver’s XML schema $S_{\text{in}}$ is contained in the sender’s XML schema $S_{\text{out}}$: $\text{partOf}(S_{\text{in}}, S_{\text{out}})$.

After identifying matching concepts, the composer connects fitting edges by a transformation activity node that defines a conversion which possibly needs to be performed in the real-time execution of the combined process. This conversion is given by the mapping function $\text{map}_{S_1 \rightarrow S_2}$. It defines how to transform actual data corresponding to the XML schema of the sender $S_1$ to a message corresponding to the receiver’s format $S_2$. The result of the composition is a business process that contains the process steps of both parties, their interconnections via mapping activities, and those inputs and outputs that could not be interconnected. The composition therefore is successful, when there are no inputs and outputs left that could not be connected to corresponding communications of the other party.

For the next step of our use case, the composer result needs to be translated into the process representation format of the enhanced process modelling tool and will thus be presented to the user. Also in the case of an unsuccessful composer execution, the partly connected business process will be fed to the enhanced process modelling tool as a first suggestion for adaptation by the user.

3.2 Implementation

We contribute a lifting engine and mapping engine to solve the mediation problems. The novel approaches rely on a domain ontology to improve lifting and mapping results.
We propose a mediation from existing WSDL files to the ontological level such that the ILOG composer [1] can be reused in order to achieve the automatic composition. Furthermore, we apply an existing SAP tool for graphical process modelling as well as a SAP process repository [6].

3.2.1 Lifting Engine

In this subsection we will describe implementation details of the Lifting Engine. As described in section 3.1.3, the goal of this engine is to align entities of an XML schema with entities of the domain ontology. This alignment is achieved by executing several independent matching algorithms and combining their individual results. The overall architecture of the Lifting Engine is depicted in figure 3.2.

![Figure 3.2: Architecture of the Lifting Engine.](image)

In order to create a alignment of an XML schema $S$ to the domain ontology $O_D$ three steps are executed by the Lifting Engine. First $S$ and $O_D$ are loaded into the engine and stored in an internal representation format. Next the set of individual matchers is executed. Each individual matcher compares each entity in $S$ to each entity in $O_D$. The results of the individual matchers are stored in the similarity cuboid. Each entry of the similarity cuboid contains a floating-point number between 0 and 1 representing the similarity between the given XML schema entity and the given ontology entity. If we assume $m$ XSD entities in $S$, $n$ entities in $O_D$ and $p$ independent matchers the similarity cuboid is of dimension $m \times n \times p$. Note that each independent matcher stores its results in a matrix of dimension $m \times n$. In the final step the results of the individual matchers are aggregated resulting in the required alignment $A_{S \rightarrow O_D}$.

**Internal XML Schema representation:**

In our framework XML schemas are represented as a node labeled tree. Starting from the root element of a XML schema a tree node is created for each schema entity. This tree node contains, besides other, the following information:
<table>
<thead>
<tr>
<th>Link Type</th>
<th>Property</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subclass</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Datatype Property</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Object Property</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 3.1: Tuple based representation of ontologies in the lifting engine.

- Link to the parent element
- List of children
- Name of the entity
- Type of its corresponding XML schema entity (i.e. Element, Complex Type, Simple Type or Attribute)
- Optional cardinality restrictions
- Optional XML schema type
- Optional documentation

This representation enables us to treat each entity of an XML schema uniformly in the framework without losing any of the information contained in the XML schema.

**Internal ontology representation:**

As we also need to represent each ontology entity in a uniform way, we decided to represent an ontology as a set of tuples. Each tuple consists of a property component of type `OWLProperty`, a range component of type `OWLClass`, and optional cardinalities. This representation enables us to treat subclass relations, object property and datatype property uniformly. A subclass relationship for example is expressed by a tuple with just a range and no property. Details on how we use these tuples to represent different types of relation in an ontology are represented are given in table 3.1.

**Individual Matchers**

In order to exploit different kinds of the information in the Lifting Engine, we have chosen a set of different elementary matching approaches. The individual matchers exploit information on the syntactic element level, the external element level and the structure level. Table 3.2 gives an brief overview of the individual matchers we have currently implemented in the Lifting Engine.

Each of the matchers takes as its input a schema entity and a ontology entity and calculates a probability that the two match based on different criteria. The Edit-Distance matcher for example uses the name of the schema entity and the name of the ontology entity and calculates the number of necessary edit operations (i.e. delete, insert and change operations). This number is then normalized to the interval [0, 1] in order to get the probability that the two entities match.

---

1. OWLProperty and OWLClass are part of the OWL API that we are reusing in our framework.
2. See [8] for a detailed explanation of this classification of elementary matching algorithms.
<table>
<thead>
<tr>
<th>Matcher Type</th>
<th>Matcher</th>
<th>Execution</th>
<th>Auxiliary Info</th>
<th>Exploited</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple</td>
<td>Edit-Distance</td>
<td>entity names</td>
<td>-</td>
<td>string-based</td>
</tr>
<tr>
<td>simple</td>
<td>Tokenizer</td>
<td>entity names</td>
<td>-</td>
<td>language-based</td>
</tr>
<tr>
<td>simple</td>
<td>Synonym</td>
<td>entity names, WordNet</td>
<td>WordNet</td>
<td>Linguistic resources</td>
</tr>
<tr>
<td>simple</td>
<td>Children</td>
<td>neighbors</td>
<td>-</td>
<td>Structure Level</td>
</tr>
<tr>
<td>simple</td>
<td>Restriction</td>
<td>cardinalities</td>
<td>-</td>
<td>Syntactic Level</td>
</tr>
</tbody>
</table>

Table 3.2: Matcher library.

Note that this is only a first selection of individual matching algorithms. In the future we plan to extend this set and especially focus on which combination of individual matchers performs best in the context of lifting.

**Aggregation**

The aggregation step can be seen as an additional hybrid matcher on top of the composite matcher. The aggregation process is controlled through the ontology. It starts at the root element of the XML schema and the corresponding ontology concept. The corresponding ontology concept is defined as the ontology concept with the highest similarity value when compared to the root element of the XML schema. Given, that the schema entity \( e_{S,1} \) and the ontology entity \( e_{O,1} \) correspond to each other, the aggregation will in the next step only investigate neighbors in the ontology as matching candidates for the children of \( e_{S,1} \).

Neighbors in the ontology are defined as concepts that are connected to \( e_{O,1} \) by either a subclass relation or an object property. Threshold values are used to limit the size of the neighborhood in the ontology (e.g., only 2 levels of subclasses are included). The selection of the candidates in the aggregation step significantly reduces the search space for possible matches. This eliminates mismatches and improves quality of the results achieved by the Lifting Engine.

### 3.2.2 Mapping Engine

This subsection describes the implementation details of the Mapping Engine. The task of the mapping engine is to create a executable mapping between two schemas \( S_1 \) and \( S_2 \). In order to create this mapping, the Mapping Engine takes two alignments \( AS_1 \rightarrow O_D \) and \( AS_2 \rightarrow O_D \) between the two input schemas and the domain ontology as an input. Based on these alignments and the domain ontology, a set of relations \( r = (\{e_{S_1}\}, \{e_{S_2}\}, e_{O_D}) \) between \( S_1 \), \( S_2 \) and \( O_D \) are calculated. Each relation describes how entities of schema \( S_1 \) are related to entities of schema \( S_2 \) based on \( AS_1 \rightarrow O_D \) and \( AS_2 \rightarrow O_D \). A simple relation \( r = (\{FirstName\}, \{GivenName\}, FirstName) \) could e.g. relate the element \( FirstName \) of schema \( S_1 \) to the element \( GivenName \) of schema \( S_2 \) based on the ontology concept \( First Name \). Next these relations are used by the Mapping Rule Generator to create the mapping necessary to transform \( S_1 \) into \( S_2 \). The mapping rules are generated based on mapping rule templates stored in an external knowledge base. The high level architecture of the Mapping Engine is depicted in 3.3.
3.2.3 Domain Ontology

The purpose of the domain ontology is to provide the domain knowledge in a machine interpretable format. In the carrier shipper scenario described in section 2 the domain ontology consist of concepts describing i) the involved parties (e.g. carriers, shipper), ii) involved business process steps (e.g. shipping, invoicing, etc.) and iii) the involved B2B messages (e.g. purchase order message, etc.) and their relations.

Currently we are in the process of developing the necessary ontology. Therefore we will present a detailed description of the involved concepts and relations in the final version of the deliverable.

3.2.4 Composition Engine

For automatically composing the collaborative business process we apply the ILog Composer [2, 3, 1]. The ILog composer provides the entire range of functionality required to make semantic Web service composition useful:

- edit composition requests by selecting goals representing target atomic Web services
- formulate a composition request by optionally specifying how input/output messages relate, possibly involving constraints explicitly formulated: for instance when some data is the aggregated sum of others (as is frequent in composition in the presence of cost data).
- run the composer to produce an orchestration satisfying the above requirements, and correct with respect to the choreographies of all participant Web services, by using the DIP discovery module or, optionally, a specialized, efficient indexing directory which is called Hotblu in this document.
- extract a choreography from the resulting orchestration

Figure 3.3: Architecture of the Lifting Engine.
• invoke a module translating choreographies and orchestrations to ASMs that can potentially be emulated by WSMX

• publish the results as a fully functional DIP SWS,

3.2.5 Graphical User Interface

We reuse an existing software-modeling tool developed by SAP Research, called Maestro, for the graphical user interface. It is a prototype for creating public process from the private processes in a company. To do so, the user first creates a copy of the private processes via a click of the mouse. Like a type of black box, the public process combines various tasks, which are grouped together on the screen. It hides various items of internal information. If a supplier is selected after a bidding process, they remain unaware of the price and scope of services offered by the next most expensive competitor for instance. The related competitor information remains hidden behind the public ordering procedure. This prevents the supplier from gaining access to any information about their competitors bids. Other public process steps within the supply chain are constituted in the same way. As this method is based on open standards, business partners do not need to use the same tools.

The solution developer combines the process steps that make up the business partners’ public processes in the correct order via mouse and GUI without doing any programming. Once all steps are completed, Maestro translates the processes into executable XML models. The result is a comprehensive B2B ordering process which hides the companies internal steps. Therefore, Maestro enables a comparatively complex (programming) task be solved with no customer-specific programming. The intelligent tool reduces complexity, accelerates business processes and their adaptation, and reduces operating costs. [7]

There is a close connection to the workflow runtime engine Nehemiah. Maestro offers the possibility to deploy process definitions directly into Nehemiah, and assign partners from the Nehemiah repository to a corresponding private process. Furthermore, it offers the possibility to assign tasks from the Gabriel repository, a task enactment tool which is introduced in the next section. [5]

Nehemiah is a workflow runtime engine that allows the execution of business processes modeled in Maestro. The process representation is based on Scalable Vector Graphics (SVG) and you can start a process via a web browser interface (compare Figure 3.4). There you can simulate a process flow by changing the activity status manually with a mouse click. Due to the help of color coding, you can keep track of the current status for each activity on the execution page. Besides starting new processes, a user can open existing instances and monitor, how far the process flow is completed. Nehemiah itself does not provide any functionality behind activities. This is the responsibility of the task enactment tool Gabriel that is described next.

3.2.6 Process Repository

A business process is as a set of logically related tasks. The Gabriel tool is responsible for defining and enacting the concept of such a task. At runtime Gabriel decides whether the current task is an automated or a manual one. In case of an automated task, Gabriel distinguishes between service tasks and user tasks. User tasks are assigned to a workflow participant, and depending on different routing rules those tasks
are placed into the work list of the responsible person. Service tasks, on the other hand, will invoke a Web service assigned to them. [6]
4 CONCLUSION

The deliverable discussed the need for combining Web service mediation and composition techniques in the frequently occurring carrier shipper scenario. Our prototype of an enhanced process modelling tool, supports automatic generation of collaborative business processes and, thus, addresses one of the major challenges of ESA. The other challenge, namely the run time discovery of carriers will be discussed in Deliverable 5.7.

After the successful implementation of this first scenario, the described setting can be extended to a more comprehensive application in a later version. In the current proposal, the composed business process is being created during design-time. When a carrier changes its conditions, the process of composition needs to be executed again in order to compute the potential adoptions to the process instance. In a more dynamic implementation, this step could be executed each time a customer requests a shipment. This way, the system would immediately and automatically incorporate changes to the carrier capabilities.
REFERENCES


