

Model-driven development of a mediation service¹

Dick Quartel^a, Stanislav Pokraev^a, Rodrigo Mantovaneli Pessoa^b and Marten van Sinderen^b

^a*Telematica Instituut*

{Dick.Quartel, Stanislav.Pokraev}@telin.nl

^b*University of Twente*

{mantovanelir, M.J.vanSinderen}@ewi.utwente.nl

Abstract

Although service-oriented architectures offer real benefits when pursuing application integration and business flexibility, there are still no satisfactory solutions for dealing with existing systems that need to cooperate while their services have no perfect match. In the case of incompatible services, a 'mediator' may be introduced which resolves (semantic) interoperability problems by intervening in the cooperation between systems. Building mediators is currently often a manual process, resulting in dedicated IT-driven solutions, with no concern for re-use of process, models or code. This paper presents a framework to guide the development of mediators, with the following objectives: (i) uncover and capture the actual interoperability problem that needs to be solved; (ii) allow the involvement of non-IT (i.e., business) experts in the development of the solution; (iii) support evolution of the solution and re-use of results in case of changing interoperability requirements; (iv) facilitate automation of parts of the process. The framework is based on service-oriented, model-driven and semantic web techniques. Available tool support for the different steps in the framework is indicated.

1. Introduction

Re-use and composability are considered as important benefits of the service-oriented paradigm. These benefits do however not come for free. Re-usable services need to be identified, specified and, possibly, re-engineered. For this purpose, standardization guidelines may be developed that reflect best-practices and put general quality principles like generality, orthogonality and parsimony into practice. Composition techniques need to be able to apply knowledge about existing services, in order to find combinations of services that match some service request, and select the best among alternatives. The idea behind

standardization of services is to facilitate the composition process by reducing the search and solution space. The realization of this idea is however difficult and takes time. Instead, the composition and integration of services from proprietary and legacy systems is currently common practice for many companies.

Over the past years, service composition has emerged as an active research area, which has resulted in various approaches and techniques ([8],[25],[15],[1]). However, the applicability of automated approaches is still limited considering the kind of assumptions being made. Furthermore, many approaches are defined at a technology level and cannot easily be used with alternative technologies.

This paper contributes to the area of service composition by presenting a framework for service mediation. We approach mediation as a service composition problem, where two or more systems have to cooperate using non-interoperable services. In order to resolve the differences between these services, a mediator is designed. Two types of mediation are considered: (i) data mediation to resolve differences between the information models being used, and (ii) process mediation to resolve differences between the interaction protocols being assumed by the systems.

Nowadays, building mediators is mostly a manual process performed by IT experts that consult business domain experts only at the requirements elicitation phase. Often, such projects fail due to miscommunication and misinterpretation of these requirements, or the resulting solutions come at a high price because of the manual labour required to build and maintain them. To address these issues we propose a framework for building mediation solutions by using model-driven and semantic web techniques. Model-driven techniques are used to lift the design of the mediator from technology to (platform-independent) model level, in order to clearly capture the semantics of the problem and proposed solution, and facilitate the involvement of business domain experts in the design process. The use of semantic web technology allows

¹ This work is part of the Freeband A-MUSE project (<http://a-muse.freeband.nl>), which is sponsored by the Dutch government under contract BSIK 03025.

one to reason about the design and aims at finding (semi-)automated techniques to construct the mediator.

This paper is further structured as follows. Section 2 analyses the mediation problem and describes an example scenario. Section 3 presents our mediation framework, including a method for composing mediators. Section 4 applies the method to the example scenario, with the composition task being performed manually. Section 5 investigates ways to automate (parts of) the composition task using semantic web technology. Section 6 discusses related work. And section 7 presents our conclusions.

2. Mediation problem and scenario

This paper addresses the problem of integrating existing systems, in particular business processes and enterprise applications. Following the service-oriented paradigm, we assume that such systems are defined in terms of the services they provide to and request from their environment, e.g., using WSDL. Furthermore, we assume these services can not be changed.

2.1. Definition and approach

Unless systems have been designed with cooperation in mind, it is unlikely that their services will match perfectly. We distinguish two types of mismatches:

- *data mismatches*, which occur when systems use different information models (vocabularies) to describe the messages that are exchanged by their services;
- *process mismatches*, which occur when systems use services that define different messages or different orderings of message exchanges.

Service mediation aims at resolving these mismatches. Webster's defines mediation as "to act as intermediary agent in bringing, effecting, or communicating" and "to interpose between parties in order to reconcile them". Correspondingly, we define service mediation as "to act as an intermediary agent in reconciling differences between the services of two or more systems". The need for an intermediary, further on denoted as mediator, is imposed by the assumption that the mediated services can not be changed. The definition abstracts, however, from whom will perform the mediator role, e.g., some of the existing systems or a 'third' system.

We approach the design of a mediator as a composition problem: each service that is requested by some of the involved systems has to be composed from one or more services that are provided by the other systems and, possibly, by the same system. This corresponds to fixed public process composition as described in [5], with the composition (integration) process acting as a mediation broker.

Figure 1 illustrates our approach for the case of two systems. Mediator M offers a mediation service that matches requested service S1 of system A by composing services S3 and S4 that are offered by system B. The mediator should provide such a mediation service for each service that is requested by systems A and B.

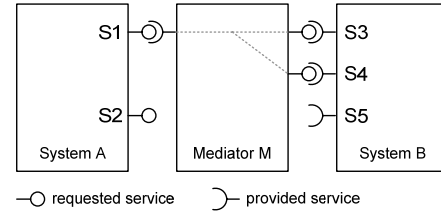


Figure 1. Service mediation as service composition

A mediation service as defined above provides interoperability for each individual service that is requested by some system. This may however not guarantee interoperability in scenario's where multiple of these requested services have to cooperate. Therefore, our approach allows one to model this cooperation and validate whether it satisfies the goals for integration.

2.2. Scenario

To illustrate our approach we present a scenario based on the Semantic Web Service (SWS) Challenge [27]. This challenge provides a standard set of problems, based on industrial specifications and requirements.

A manufacturing company called *Moon* uses two back-end systems to manage its order processing: a *Customer Relation Management (CRM)* system and an *Order Management (OM)* system. *Moon* has signed an agreement with a customer, called *Blue*, to exchange purchase order messages in *RosettaNet PIP 3A4* format. Currently, the back-end systems of *Moon* use a proprietary data model and interaction protocol that differ from the ones used by RosettaNet. Figure 2 depicts the scenario.

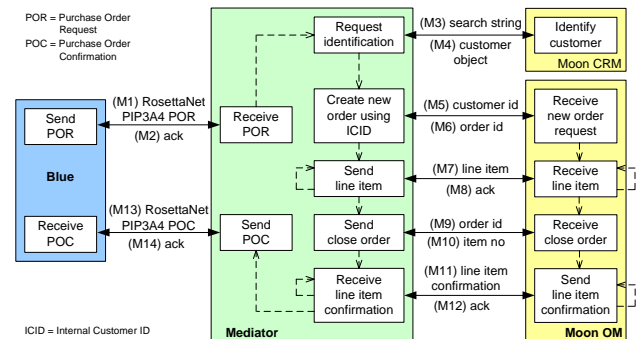


Figure 2. Mediation scenario

The interaction between both systems is initiated by Blue who sends a PIP 3A4 *Purchase Order Request*

message (M1). PIP 3A4 enables a buyer to issue a purchase order and to obtain a quick response from the provider that acknowledges which of the purchase order product line items are *accepted*, *rejected*, or *pending*. For brevity we only show excerpts of some messages, in this case of M1:

```
<?xml version="1.0" encoding="UTF-8"?>
...
<fromRole>
  <PartnerRoleDescription>
    <ContactInformation>
      <contactName>
        <FreeFormText>Mr Blue</FreeFormText>
      </contactName>
    </ContactInformation>
  </PartnerRoleDescription>
</fromRole>
...
<PurchaseOrder>
  <ProductLineItem>
    ..
    <LineNumber>1</LineNumber>
    <OrderQuantity>
      <requestedQuantity>
        <ProductQuantity>1</ProductQuantity>
      </requestedQuantity>
    </OrderQuantity>
    <ProductIdentification>
      <GlobalProductIdentifier>00614141000012</GlobalProductIdentifier>
    </ProductIdentification>
    <requestedUnitPrice>
      <FinancialAmount>
        <GlobalCurrencyCode>EUR</GlobalCurrencyCode>
        <MonetaryAmount>1067.54</MonetaryAmount>
      </FinancialAmount>
    </requestedUnitPrice>
  </ProductLineItem>
</PurchaseOrder>
```

The Purchase Order Request message must be synchronously confirmed by an *Acknowledgement of Receipt* message (M2). According to the RosettaNet standard a Purchase Order Request is sent using a single message. However, in order for Moon to be able to process a purchase order, several steps have to be made.

First, the customer needs to be identified by sending a search string to Moon's CRM system (message M3),

```
<SearchCustomer>
  <searchString>Blue Company</searchString>
</SearchCustomer>
```

which replies by sending a customer object that matches the search string (message M4).

```
<SearchCustomerResponse>
  <customerId>1</customerId>
  <roleCode>Buyer</roleCode>
  <contactName>Mr John Smith</contactName>
  <email>john@example.org</email>
  <telephone>+31 1234567890</telephone>
  <businessName>Blue Company</businessName>
  <postalCode>1234AB</postalCode>
  <city>Amsterdam</city>
  <street>Blue street 12</street>
  <countryCode>Netherlands</countryCode>
</SearchCustomerResponse>
```

Next, the creation of a new order is requested by sending the customer id (message M5) to Moon's OM system, which returns the id of the newly created order (message M6). After a new order is created, Moon's OM system expects all order lines to be added one by one (message M7).

```
<addLineItem>
  <LineItem>
    <orderId>123</orderId>
    <item>
      <articleId>456</articleId>
      <quantity>1</quantity>
    </item>
  </LineItem>
</addLineItem>
```

These messages are acknowledged synchronously (message M8) by sending the order id and an item id. Once all order lines have been added, Moon OM is requested to close the order (message M9), and returns the number of items that has been received (message M10). Subsequently, Moon's OM system confirms the status of each order line (message M11), which is acknowledged synchronously (message M12) by the mediator.

After all order lines have been confirmed a RosettaNet PIP3A4 Purchase Order Confirmation message (M13) is sent to Blue and confirmed synchronously by an Acknowledgement of Receipt message (M14).

3. Mediation framework

As part of the A-Muse project [2], a mediation framework is being developed to support the design, implementation and validation of mediation services. This framework consists of the following elements:

- a conceptual framework for modelling and reasoning about services, called COSMO [22];
- languages to express service models using COSMO, which currently include ISDL [11],[24], OWL [14], SPARQL [20] and Java;
- techniques to analyse the interoperability and conformance of service models [23];
- transformations from service design to service implementation level [7], [21], and vice versa;
- tools supporting the editing, analysis and transformation of service models [21]; and
- a method for developing mediation services.

This paper focuses on describing and illustrating our method for service mediation. The method uses and relates the other elements of the framework listed above, which are only explained here as far as required for a proper understanding of the paper. This includes a brief description of how services are modelled in the second part of this section.

3.1. Method

Figure 3 illustrates the steps that constitute our method for service mediation. For convenience, the integration of two systems is considered, but the same steps apply to the case of multiple systems.

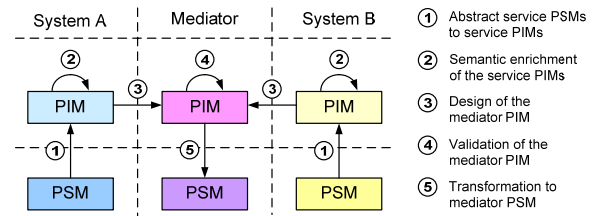


Figure 3. Method for service mediation

In general, the services of systems that have to be integrated are described at implementation (technology) level, e.g., using WSDL. The method starts with “lifting” these service descriptions to design level, by abstracting from implementation specific information. Such information may unnecessarily complicate the design of an integration solution, and therefore hinder the participation of business domain experts that are knowledgeable about the integration requirements at business level, but don’t (want to) know how these requirements are implemented at IT level. In terms of the MDA (Model Driven Architecture) this means that we transform the service PSMs (Platform Specific Models) of the systems being integrated to their respective service PIMs (Platform Independent Models).

Subsequently, the service PIMs may be semantically enriched by adding information that could not be derived (automatically) from the service PSMs. For example, a service PSM may be complemented with some text document that describes part of the service in natural language. Alternatively, interviews or even code inspection may be used to obtain information that is missing from the service PSMs. The purpose of semantical enrichment is to make models precise and complete, which in turn is necessary to enable formal reasoning about and, potentially, the (semi-) automated generation of the integration solution.

The next steps represent the design, validation and implementation of the integration solution, i.e., the mediator PIM. The design step can be split into two parts: (i) the design of an information model, and (ii) the design of a behaviour model for the mediator. The purpose of the information model is to enable data mediation, by defining a mapping between the vocabularies of the systems being integrated. The purpose of the behaviour model is to enable process mediation by defining a mapping between the services that are requested and the services that are provided by the systems being integrated (see section 2.1).

The validation step is used to analyse whether interoperability is obtained by the proposed integration solution. This step could be omitted in case one would support the automated composition of mediators. But for now this seems an ideal that can not be realized yet.

In the final step, the mediator PIM is transformed to an implementation, the mediator PSM.

3.2. Service modelling

We define a *service* as the establishment of some effect (or value) through the interaction between two or more systems. The COSMO framework defines concepts to support the modelling, reasoning and analysis of

services. These concepts are structured along three axes as depicted in Figure 4.

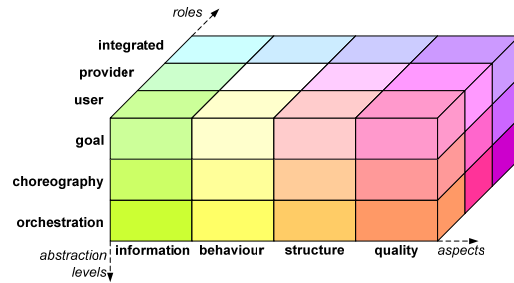


Figure 4. The COSMO framework

The horizontal axis distinguishes four aspects, i.e., *information*, *behaviour*, *structure* and *quality*, representing categories of service properties that need to be modelled. This classification corresponds to aspects found in frameworks for enterprise architectures like GRAAL [9] and ArchiMate [13].

The vertical axis distinguishes three global abstraction levels at which a service can be modelled:

- a *goal* models a service as a single interaction, where the interaction result represents the effect of the service as a whole;
- a *choreography* refines a goal by modelling a service as a set of multiple related, more concrete interactions;
- an *orchestration* implements a service using a central coordinator that invokes and adds value to one or more other services.

We note that these abstraction levels should not be treated as absolute levels, but can again be considered in more or less detail, resulting in sub-levels of abstraction.

The diagonal axis distinguishes the roles of the systems involved in a service: the *user*, *provider* and *integrated role*. The integrated role abstracts from the distinction between a user and provider by considering interactions as joint actions, thereby focusing on what the user and provider have in common.

This paper mainly considers choreographies and orchestrations from the behaviour and information aspect, and by distinguishing between a user and provider role. Furthermore, services are modelled close to the level at which they are described using WSDL, while abstracting from technology details. Therefore, and for brevity, we only explain COSMO’s operation concept below and its notation using ISDL. For an explanation on concepts supporting the modelling and design of services at more abstract levels we refer to [22].

Figure 5(i) and (ii) depict the operation concept and its interpretation in terms of a flow chart-like notation, respectively. An operation represents a composition of three instances of message passing: the sending (invoke)

and receipt (accept) of an invocation, followed by either the sending (reply) and receipt (return) of the invocation result, or the sending (fault) and receipt (catch) of a fault message. The use of the reply-return and the fail-catch message passing instances are optional, i.e., either one or both parts may be omitted; e.g., to model one-way operations.

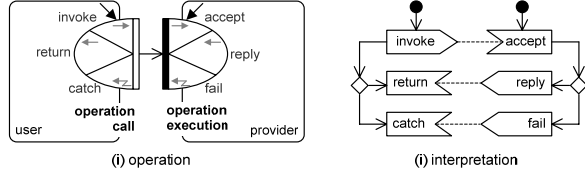


Figure 5. Operation concept

Figure 6 depicts an example ISDL model of an order handling choreography consisting of four operations (inspired by the scenario of section 2.2): create represents the creation of an order, addItem represents adding an item to the order, close represents closing the order, and confirmItem represents the confirmation of the status of an added item. Operations addItem and confirmItem are modelled as repetitive operations (represented by double border lines). A textbox defines the parameters associated with an operation, including the constraints on these parameters (between square brackets). Constraints can be preceded by the symbols “1.” or “+.” to distinguish between constraints for the first or next occurrences of an operation repetition, respectively. The parallelogram shaped boxes define behaviour variables, also called behaviour items. For example, behaviour Customer defines two variables: o representing the order to be sent to the retailer, and id representing the id of the current order line item being sent. Each occurrence of operation addItem sends the next order line item (represented by parameter it and constraint $it = o.getItem(id)$), and receives the line item id as acknowledgement in return. For this purpose, the value of variable id must be 0 for the first occurrence and is increased by 1 for each next occurrence. The retailer adds each received item to its own behaviour variable o as represented by constraint $o.putItem(id, it)$.

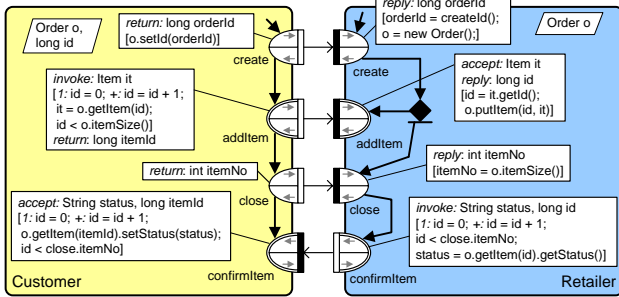


Figure 6. Example choreography

The repetition of addItem terminates once all line items have been added, as represented by constraint $id < o.itemSize()$, upon which the customer wants to execute operation close. Since the retailer does not know when all line items have been added, a disabling relation (represented by a black diamond on top of a horizontal bar) is used to model that it is willing to execute both the close and addItem operation after an order has been created, but the occurrence of new addItem operation instances is disabled (disrupted) as soon as the close operation occurs.

4. Application of the framework

This section illustrates the application of the mediation framework to the example scenario of section 2.2. For this purpose, the method of section 3 has to be made more concrete by deciding on, amongst others, the type of PSMs that are considered, the languages to be used at PIM level, and related to these choices the transformations and analysis techniques that are needed, c.q. have to be developed. This means that in time the mediation framework may be populated with different instances of the mediation method, depending on the type of integration problems that have been addressed.

4.1. Step 1: Abstract from PSMs to PIMs

In this step, we derive the platform independent information and behaviour models of the services of Blue and Moon, which are specified by WSDL documents. Figure 7 illustrates this step. The behaviour models are represented using ISDL, and the information models are specified using a combination of UML class diagrams (for visualization) and Java (for execution).

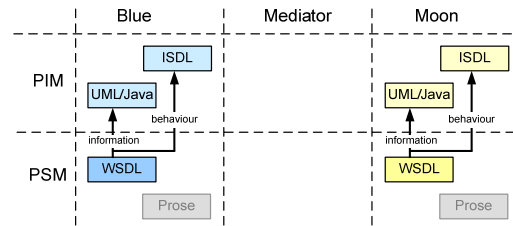


Figure 7. Abstract from PSMs to PIMs

This step is automated using the WSDL import function of the Grizzle tool [11]. This tool provides an integrated editor and simulator for ISDL, and uses Java to represent and execute operation parameter constraints (see section 3.2). The WSDL import function enables a user to import a WSDL specification by providing the URL of this specification. The user can choose to either import a single operation, single port type or the complete WSDL definition. Furthermore, the user may choose whether the web service should be considered from a

client or server perspective. Accordingly, a behaviour model is generated that represents the user (client) or provider (server) role of the web service, in terms of operation calls or operation executions, respectively. In addition, an information model is generated consisting of Java classes that represent the information types that are referred to by the operations in the behaviour model. The transformation of WSDL to ISDL and Java is implemented using JAXB and JAX-WS ([12]). The EclipseUML tool ([10]) is used to visualize and manipulate the information model using UML class diagrams.

As an example, Figure 8 depicts the ISDL behaviours of Blue and Moon's OM system that are generated from the corresponding WSDL descriptions. Besides the operation parameters, the text box of operation `createNewOrder` shows the stereotype information that is added to the operation definition. This information can be used to execute the modelled web service as part of the simulation of an ISDL model (see section 4.4).

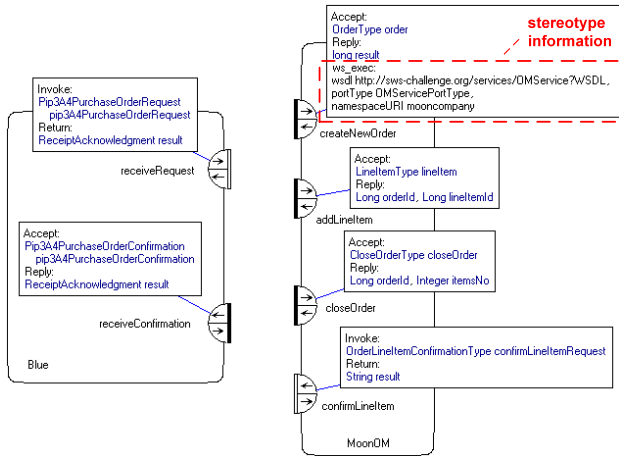


Figure 8. ISDL generated for Blue and Moon OM

4.2. Step 2: Semantic enrichment of PIMs

The WSDL descriptions of the example scenario define the services that are provided by Blue, Moon and the Mediator, in terms of their operations and the types of the input and output messages of these operations. However, WSDL does not define the interaction protocols, i.e., the possible orderings of the operations. Therefore, to derive the complete PIMs of Moon and Blue, we have to use and interpret the textual descriptions that are provided with the integration case (the boxes labelled "Prose" in Figure 3). This is a manual process.

Firstly, the behaviour models that were generated in step 1 are completed by defining relations between operations. These relations can be derived from the scenario description. This includes the explicit modelling of the "loops" in the schema of Figure 2, representing the

repetitive process of adding and confirming line items. Figure 9 depicts the enriched model of the service requested by Blue and the service provided by Moon OM.

Secondly, the information model may be enriched by interpreting the scenario description. A WSDL description defines the syntax of the messages that are exchanged, but provides no information about their semantics. This semantics can be made explicit by defining new classes and use these classes to relate the existing (generated) classes. Furthermore, the meaning of classes and their properties may be defined by a mapping onto some domain-specific ontology, e.g., the Universal Data Element Framework [28]. The benefits of these types of semantical enrichment can however only be fully exploited when using a language that allows one to explicitly model and reason about the semantics of classes and their properties. Section 5 introduces and explores the use of such a language, i.e., OWL, to represent information models.

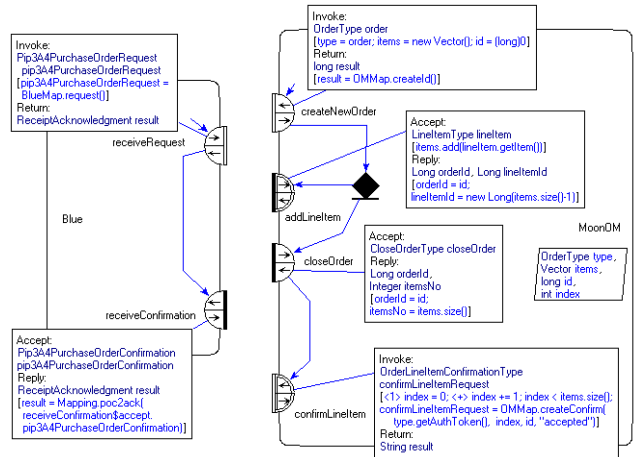


Figure 9. Enriched models of Blue and Moon OM

4.3. Step 3: Design of the mediator PIM

In this step we design the behaviour and information model of the Mediator.

The information model of the Mediator is constructed from the union of the information models of Blue and Moon. For the same reason as explained at the end of the previous section, this information model is not enriched to define the relationships between the classes and properties from the information models of Blue and Moon, except for informal annotations that may explain these relationships using natural language. Section 5 discusses a formal approach in defining such relationships using OWL. The information model is extended, however, with classes to represent status information of the Mediator, such as the set of order line items that have been confirmed so far.

The construction of the behaviour model of the Mediator requires the definition of:

1. the services provided and requested by the Mediator;
2. the composition of these services by relating the operations of the services;
3. the data transformations among the parameters of the operations.

Step 1: Provided and requested services. In the example scenario, the Mediator provides one service that must match the service requested by Blue. The service provided by the Mediator can initially be defined as the 'complement' of the service requested by Blue. The complement of a service is obtained by changing each operation call into an operation execution, and vice versa, while keeping the same parameters (see Figure 10). In addition, the relations among the operations and the parameter constraints may (initially) be retained. Analogously, the services that are requested by the Mediator can be obtained by taking the complement of the services that are provided by Moon. Figure 10 depicts the resulting skeleton of the Mediator.

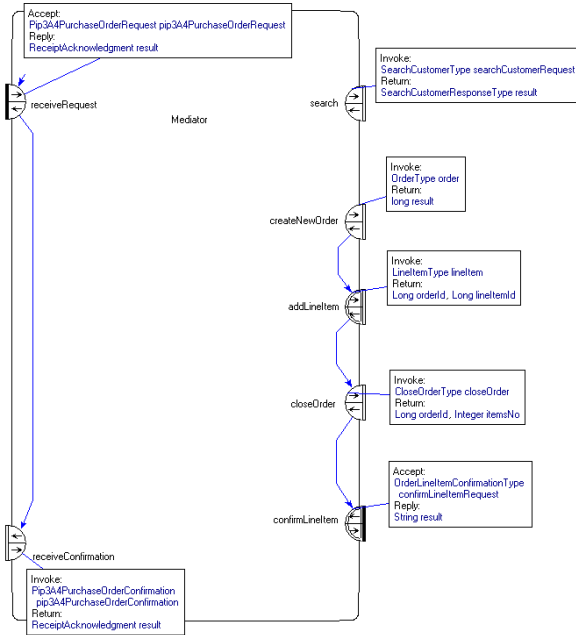


Figure 10. Skeleton of Mediator

The retained relations and parameter constraints may be refined in the next design steps, respectively. For example, the relation between operations `receiveRequest` and `receiveConfirmation` has to be implemented by the orchestration of the services of Moon. As another example, the disabling relation between `addLineItem` and `closeOrder` has already been replaced by an enabling relation, since the order should be closed only after all line items have been added (cf. Figure 6).

Step 2. Composition of services. The design of the Mediator behaviour can now be approached as the search for a composition of the requested services that conforms to the provided service. The structure of this composition is defined by the (causal) relations among the operations. Most of these relations can be found by matching the input information that is required by each operation to the output information that is produced by other operations. For example, operation `search` of Moon's CRM service requires as input a search string that can be matched to some element of the customer information that is part of the purchase order information received by operation `receiveRequest`. This implies that a relation should be defined between `receiveRequest` and `search` (see Figure 11).

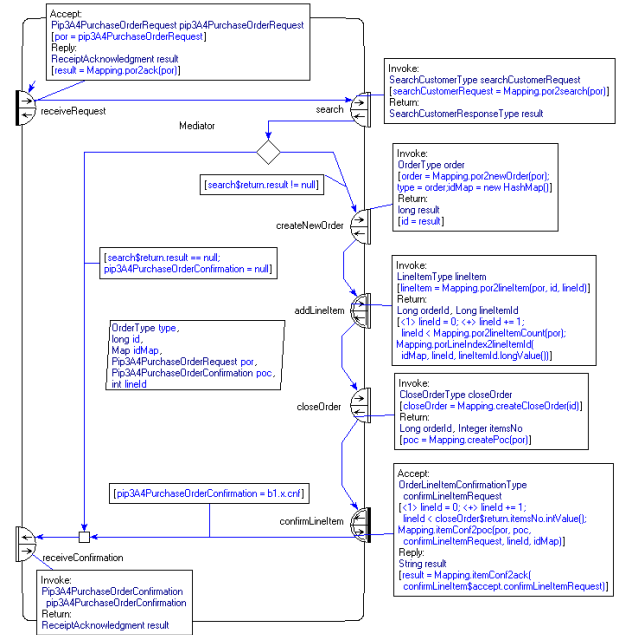


Figure 11. Design of the mediator

Matching input and output information is however insufficient to find all relations. For example, although both operations `receiveRequest` and `search` provide information that matches the input required by operation `createNewOrder`, the information that is provided by `receiveRequest` should be used. This hidden assumption has to be made explicit in the behaviour model.

Furthermore, specific processing logic may have to be designed manually. For example, the process of receiving confirmations from Moon's OM system depends on information from operations `receiveRequest` (the items to be confirmed), `createNewOrder` (the order id) and `addLineItem` (the item id used by Moon), and depends on internal status information of the Mediator, i.e., the knowledge that operation `closeOrder` has occurred and the set of confirmations that has been received so far. Even when these information requirements are given, the

relations involved in the repetitive processing of confirmations can not be derived easily, and have to be designed explicitly.

Step 3: Data transformations among parameters. The definition of the data transformations among operation parameters can be approached as a refinement of the relations among operations defined in the preceding step. These relations define for each operation on which other operations it depends, and therefore which output parameters can be referred to (i.e., used) in the generation of its input parameters. The data transformations then define how the value of each input parameter is generated from the values of the output parameters and, possibly, some internal state information of the Mediator. This involves the definition of translations between the vocabularies used by Blue and Moon. However, these translations only need to address those parts of the vocabularies that are related via the relations defined in step 2. For example, all data transformations of the Mediator have been defined in a class called Mapping. The data transformation between operations `receiveRequest` and `search` has been defined by method `por2search()` as described in the text box associated with operation `search`. This method gets as argument the value of behaviour variable `Pip3A4PurchaseOrderRequest por`. This value is assigned after operation `receiveRequest` has received the purchase order request from Blue.

4.4. Step 4: Validation of the mediator PIM

In this step, the design of the Mediator is validated by means of the following analyses:

- assessment of the interoperability between the services of Blue, the Mediator and Moon;
- simulation of the interacting behaviour of these services.

Interoperability assessment. A method for interoperability assessment has been presented in earlier work [23]. This method consists of two steps. The first step checks whether each individual interaction can establish a result. This check is based on the abstract interaction concept of COSMO, which allows complex negotiations to be modelled in which the involved systems may define their own, possible conflicting, constraints on the interaction result. In this case, however, the interactions are operations, which have been designed such that the parameter types at the sending and receiving side are the same, and the parameter values are completely determined by the sending side.

The second step checks whether the service composition as a whole can establish a result. For this purpose, the interacting behaviour among Blue, the

Mediator and Moon is viewed from an integrated perspective, where operations are viewed as joint actions, and subsequently transformed to a Coloured Petri Net. From this net we construct the corresponding occurrence graph to perform reachability analysis, using the CPNTools [6]. This analysis allows us to check whether operations can be reached, and in a certain order.

Simulation. The simulation of ISDL behaviours is supported by the Grizzle tool [11]. Simulation allows a designer to analyse the possible orderings of operations occurrences, as well as the information results that are established in these operations. In addition, the Grizzle simulator provides hooks in the simulation process to execute application code upon execution of an operation. This enables us to perform real web service invocations and incorporate the results that are returned by web services during the simulation. For this purpose, stub-code is linked to a modelled web-service operation *call*. This code is generated automatically based on stereotype information that has been retained during the WSDL import (see Figure 8), such as the web service's end-point address and port type name.

Furthermore, the simulator allows external web-clients to invoke a modelled web-service operation *execution* (see Figure 5(i)). A web service proxy is automatically generated and deployed in an application server, again using forementioned stereotype information. This proxy is responsible for handling the reception of the invocation request and the return of the invocation result. In between, the proxy delegates the calculation of the invocation result to the simulator, which indicates to the user that the operation is enabled and waits till the user requests the simulation of this operation.



The support for real, also called 'live', web service invocations, allows one to use the simulator as an orchestration engine in which an orchestration can be executed by simulating its ISDL model. This means that that the simulator provides, in principle, an implementation for the Mediator. However, this simulator does not support important properties of an execution environment, such as performance, monitoring, etc. Therefore, we transform the Mediator design towards a BPEL process in the next step.

4.5. Step 5: Derivation of the mediator PSM

In this step, an implementation is derived for the Mediator design. For this purpose, a transformation has been developed that transforms an orchestration model in ISDL to a BPEL specification that can be executed on a standard BPEL engine. This transformation consists of two main tasks:

1. the recognition of common behaviour patterns, such as workflow patterns, and their translation to a composition of the following basic patterns: *sequence*, *concurrency*, *selection* and *iteration*;
2. the realization of these basic patterns using the BPEL constructs *bpel:sequence*, *bpel:while*, *bpel:flow* and *bpel:if*, respectively.

In addition, the Mediator model has to be annotated with information that is required as input to the transformation. This information concerns choices in the mapping of abstract ISDL behaviour constructs onto concrete BPEL constructs or extra design information that is needed at platform specific level. Figure 12 depicts these annotations, which are added as stereotype information to ISDL model elements. The ‘process’ annotation is used to indicate which behaviour definition represents the complete orchestration behaviour and thus has to be mapped onto a BPEL process. The other annotations deal with the mapping of operations and the information model onto WSDL. A transformation has been implemented that generates a complete BPEL/WSDL model from a properly annotated ISDL model. For more information we refer to [7], [21].

Model element	Annotation
	process targetNamespace schemaNamespace schemaLocation
	partnerLink portType operation namespace*

* only for operation call

Figure 12. BPEL annotations

5. Automated support for mediation design

In this section we discuss the potential benefits of using OWL-DL [14] as a language to represent information models. OWL-DL allows one to reason about relationships between classes, properties and individuals in an information model, such as inheritance, equivalence, transitivity, etc. This support for reasoning helps in defining and analysing the mapping between information models, and assessing the interoperability between services. Furthermore, reasoning may enable the development of automated support for designing the mediator PIM.

5.1. Information model mapping

A limitation of using Java and UML class diagrams to represent information models is that these languages do not allow one to formally model and reason about the semantical relationships among classes and their properties (except for the inheritance relation). Instead,

OWL provides a number of constructs to semantically relate classes and properties:

- *equivalence*: the constructs `owl:equivalentClass` and `owl:equivalentProperty` are used to state that a particular class or property is equivalent to another class or property, respectively;
- *specialisation*: the constructs `rdfs:subClassOf` and `rdfs:subPropertyOf` are used to state that a particular class or property has a more specific meaning than another class or property, respectively;
- *disjointness*: the construct `owl:DisjointWith` is used to state that two classes cannot share instances.

Furthermore, OWL provides a number of ways to construct complex classes using `owl:unionOf`, `owl:intersectionOf` and `owl:complementOf`, or by restricting the values of a property. This way, a new class can be defined in terms of classes and properties from one information model and then asserted to be `owl:equivalentClass`, `rdfs:subClassOf` or `owl:DisjointWith` a class from another information model. An example of such a mapping is shown in Figure 13. The presented mapping relates all individuals of class `Item` that have property `status` equal to “accepted” to the class `AcceptedItem`.

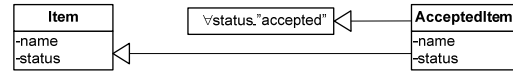


Figure 13. Mapping using restriction

In addition, OWL provides ways to define new properties for classes in one information model and then assert them to be `owl:equivalentProperty`, `rdfs:subPropertyOf` or `owl:DisjointWith` a property from another information model. Figure 14 shows an example of such a mapping.

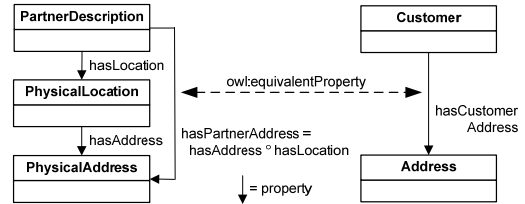


Figure 14. Mapping of equivalent properties

The main benefit of using a language like OWL in defining the mappings between the information models of Blue and Moon, is that it allows one to reason about the consistency and implications of the defined mappings. For example, some mapping may result in different sub-class relationships than expected, e.g., a class being a sub-class of the empty class, meaning that this class can never have any instances. As another example, one may want to check whether the union of the class of “rejected” items and the class of “accepted” items is equivalent to the class of “confirmed” items, and a sub-class of the class of “added” items. Furthermore, the reasoning capabilities of

OWL help in finding and revealing new mappings, since indirect relationships that are implied by a (combination of) mapping(s) will be made explicit.

In spite of this benefit, the actual data transformations between operation parameters at instance level still have to be defined. These data transformations can not be generated from the class and property mappings automatically. Currently, we investigate the use of SPARQL for this purpose, since it can query OWL ontologies and therefore exploit its reasoning capabilities at runtime. To define the information models of Blue and Moon, and the mappings between them, we have used Protégé [19] as an editing tool and Pellet [18] as a reasoner. For the derivation of information models from WSDL documents, we have used the Gloze tool [3].

5.2. Interoperability assessment

The use of OWL also facilitates the first step of the method for interoperability assessment discussed in section 4.4. In general, the systems that are involved in an interaction, or operation, may use different information models to define the type of interaction result and the constraints on the possible instances of this type that can be established. For example, Figure 15 depicts a simplified goal model [22] (see also section 3.2) of the interaction between Blue and Moon. The use of OWL to define the relations between the classes and properties in the information models of Blue and Moon, as explained in the preceding section, allows one to use a reasoner to assess whether a common interaction result can be established. This is done by checking whether the intersection of the class of interaction results allowed by Blue and the class of interaction results allowed by Moon CRM and Moon OM is satisfiable, i.e., can have instances or not. Given the relations defined in the bottom part of Figure 15, one may conclude that a common interaction result is indeed possible, consisting of a registered customer and an order having a delivery period of 3 (say days) and a price between 1000 and 2000 (say euro). Note that interoperability assessment at goal level can be useful to check beforehand if a mediation solution can be constructed without changing the existing services of Blue and Moon.

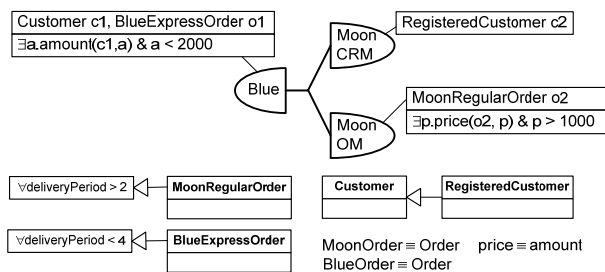


Figure 15. Example goal model

We refer to [23] for a detailed explanation on how the method for interoperability assessment in combination with OWL, can be used at different abstraction levels during a service design process.

5.3. Semi-automated mediator composition

The benefits of using OWL described above mainly concern analysis tasks. In our current work, we explore the potential of OWL in supporting the semi-automated construction of mediation solutions. In particular, we are developing the following techniques to automate parts of the mediator design.

Matching input and output parameters. The matching of input and output information as used in step 2 of section 4.3 can be automated using OWL reasoners. Given some input and output class that represent the type of input and output information, respectively, one can check whether both classes are equivalent or one is a sub-class of the other. In the former case and in case the output class is a sub-class of the input class, the output information can be used as input to some operation. However, in case the input class is a sub-class of the output class, some restriction or guard may have to be imposed on the output information before it can be used as input to the operation.

Derivation of relations between operations. Based on the automated matching of input and output parameters, a search algorithm can be developed to find the output parameters that provide the information that is required by some input parameter. The basic idea is to try in a first step to match the required input class to the classes of all output parameters. If no or only a partial match is found, this step is applied recursively to the properties of the input and/or output classes. From the resulting set of potentially partial and alternative matches, a selection can be either made manually or be proposed automatically by the algorithm based on some heuristics. Given the selected (partial) match, the (ordering) relations among the involved operations can be derived automatically.

Derivation of data transformations. Based on the selected match, also the signature for the required data transformation can be obtained automatically. For example, from the relation between the output parameter of operation `receiveRequest` and the input parameter of operation `search` as explained in section 4.3 the signature of method `por2search()` can be derived, i.e., `String por2Search(Pip3A4PurchaseOrderRequest por)`.

Integration of ISDL and OWL/SPARQL. ISDL allows bindings with different information modelling languages. Currently, this binding is being implemented for OWL/SPARQL to support simulation and transformation to BPEL. For this purpose, the simulator is

linked to an OWL reasoner that contains the information model and state associated with the simulated behaviour. During simulation, data transformations expressed in OWL and SPARQL are delegated to this reasoner. In the mapping to BPEL these data transformations are delegated to a special web-service that enables the BPEL engine to access the OWL reasoner.

6. Related and future work

Several approaches and solutions have been proposed within the SWS challenge. Here we briefly discuss the approaches based on the WSMO, SWE-ET and jABC/jETI frameworks.

The DERI approach [16] follows the Web Services Modelling Ontology (WSMO) framework. It consists of four main components – ontologies, goals, web services and mediators. The main difference between WSMO and our work is that our framework has less concepts while providing comparable expressive power. Both solutions, however, differ with respect to the way of process modelling. WSMO describes the mediator interaction behaviour by means of Abstract State Machines. A state is described by a WSMO ontology, the domain ontology constitutes the underlying knowledge representation and each transition rule defines a state transition where the condition is defined as an expression in logic, which must hold in a state before the transition is executed. For the purposes of the SWS Challenge, the provided solution assumes that the invocation order is unimportant. This is not the case though: the operations of system Moon should be invoked in a particular order.

The joint team of Politecnico di Milano and CEFRIEL [4] focuses more on the modelling of the mediator's internal logic, which is defined by a BPMN model. A coarse WebML skeleton is automatically generated from the BPMN model and manually refined by the designer. The WebML process model, specified as a graph of (web)pages, differs quite significantly from our approach. Pages consist of connected units, representing the publishing of atomic pieces of information, and operations for modifying the underlying data or performing arbitrary business actions. Units are connected by links, to allow navigation, parameter passing, and computation of the hypertext from one unit to another. The method was not natively meant to face mediation problems, but showed to adapt rather well to this class of problems.

The jABC/jETI solution [26] uses SLGs (Service Logic Graphs) as choreography models, allowing the designer to model the mediator in a graphical high level modelling language by combining reusable building blocks into (flow-)graph structures. These basic building blocks are called SIBs (Service Independent Building

Blocks) and the development process is supported by an extensible set of plug-ins that provide additional functionality. The jABC framework originated in the context of the verification of distributed systems and provides explicit support for model checking, which allows automatically proving global compliance constraints on the business logic of an SLG.

In general, service composition and mediation have emerged as an active and productive research area. Various approaches and techniques have been presented, such as static vs. dynamic, model-driven, declarative, automated vs. manual, context-based, and workflow vs. planning approaches ([8],[25],[15],[1]). In our current and future work, we investigate the use of existing AI planning techniques [17] for automatic construction of the behaviour of the Mediator. In particular, we currently focus on the use of backward-chaining techniques to discover causal relations among the activities performed by the Mediator. In our approach, we start with the activities that send messages and recursively search for activities that provide the information required to construct these messages. The search is performed using the mappings defined in the information model of the Mediator.

7. Conclusions

In this paper, we have presented a framework for developing mediation services as a means to integrate non-interoperable systems. The framework combines model-driven, service-oriented and semantic web techniques. Model-driven techniques are used to lift the design of a mediation solution from technology to (platform-independent) model level, in order to clearly capture the semantics of the integration problem and proposed solution, and facilitate the involvement of business domain experts by abstracting from implementation details. Following the service-oriented paradigm, the systems that have to be integrated are assumed to be defined in terms of the services they provide to and request from their environment. The integration problem is then approached as a service composition problem, where a mediator must be found that orchestrates and enhances the existing services provided by one system in such a way that it matches the service requested by another system.

A method has been presented to guide the development of a mediator. Tool support is provided for each of the steps in this method, including the modelling and 'live' simulation of the mediation solution, and transformations between model and implementation level. Web services (WSDL and BPEL) are assumed as implementation technology.

Currently, the composition of the mediation solution is mainly a manual process. The use of semantic web technology, based on OWL, enables automated reasoning about the mediator design, in particular the information modelling part. We have applied this in the development of a general technique to assess the interoperability of systems, and have used it in this work to validate the interoperability that is offered by the mediator.

Further, we have discussed techniques based on OWL to automate parts of the composition process of the mediator. Our ongoing and future work will focus on the elaboration of these techniques, and the development of tool support to make them practically applicable. Here, automation is not a goal in itself. In fact, we do not think semantic web technology can be used (yet) to develop fully automated techniques for building mediators. However, semantic web technology can be helpful in automating techniques that support the designer in re-using design information that is present in existing models. For example, using our mediation framework it should be easy for a developer to cope with changing integration requirements. The SWS challenge addresses this issue explicitly by requiring changes to the mediation scenario described in this paper. Our framework limits the impact of these changes to the design step of our method, such that the designer only needs to adjust the information and behaviour model of the mediator in order to generate automatically a new implementation of the mediator that reflects the changed requirements.

References

- [1] Alamri A, Eid M and El Saddik A. Classification of the State-of-the-art Dynamic Web Services Composition. In: *International Journal of Web and Grid Services*, Vol. 2, No. 2, 2006, pp. 148-166.
- [2] A-Muse project. <http://a-muse.freeband.nl>
- [3] Battle S. Gloze: XML to RDF and back again. In: *Proceedings of 2006 Jena User Conference*, 2006. <http://jena.hpl.hp.com/juc2006/proceedings.html>
- [4] Brambilla M, Celino I, Ceri S, Cerizza D, Della Valle E and Facca F. A Software Engineering Approach to Design and Development of Semantic Web Service Applications, In: *Proceedings of the 5th International Semantic Web Conference (ISWC 2006)*, LNCS 4273, 2006, pp. 172-186.
- [5] Bussler C. Semantic Web Services: Reflections on Web Service Mediation and Composition. In: *Proc. of the Fourth Int. Conf. on Web Information Systems Engineering (WISE)*, 2003, p. 253.
- [6] CPNTools - Computer Tools for Coloured Petri Nets. <http://wiki.daimi.au.uk/cpntools/cpntools.wiki>
- [7] Dirgahayu T, Quartel D and van Sinderen M. Development of Transformations from Business Process Models to Implementations by Reuse, In: *3th International Workshop on Model-Driven Enterprise Information Systems*, 2007, pp. 41-50.
- [8] Dustdar S and Schreiner W. A survey on web services composition. In: *International Journal of Web and Grid Services*, Vol. 1, No. 1, 2005, pp. 1-30.
- [9] van Eck P, Blanken H, Wieringa R. Project GRAAL: Towards Operational Architecture Alignment. *International Journal of Cooperative Information Systems* 13(3), 2004, pp. 235-255.
- [10] EclipseUML. <http://www.eclipsedownload.com/>
- [11] ISDL. <http://ctit.isdl.utwente.nl>
- [12] JAX-WS and JAXB. <http://java.sun.com/webservices/technologies/index.jsp>
- [13] Jonkers H, Lankhorst M, van Buuren R, Hoppenbrouwers S, Bonsangue M, van der Torre L. Concepts for Modelling Enterprise Architectures. *International Journal of Cooperative Information Systems*, vol. 13, no. 3, 2004, pp. 257-287.
- [14] McGuinness D and van Harmelen F. *OWL Web Ontology Language Overview – W3C Recommendation 10 February 2004*. <http://www.w3.org/TR/owl-features/>.
- [15] Milanovic N and Malek M. Current Solutions for Web Service Composition. In: *IEEE Internet Computing*, Vol. 8, No. 6, 2004, pp. 51-59.
- [16] Mocan A, Moran M, Cimpian E and Zaremba M. Filling the gap - extending service oriented architectures with semantics. In: *IEEE International Conference on e-Business Engineering (ICEBE)*, 2006, pp. 594-601.
- [17] Peer J. Web service composition as AI planning - a survey. Technical report, Univ. of St. Gallen, Switzerland, 2005.
- [18] Pellet. <http://pellet.owldl.org/>
- [19] Protégé. <http://protege.stanford.edu/overview/protege-owl.html>
- [20] Prud'hommeaux E and Seaborne A. *SPARQL Query Language for RDF - W3C Proposed Recommendation 12 November 2007*. <http://www.w3.org/TR/rdf-sparql-query/>.
- [21] Quartel D, Dirgahayu T and van Sinderen M. Model-driven design, simulation and implementation of service compositions in COSMO. To appear in: *Int. J. of Business Process Integration and Management*.
- [22] Quartel D, Steen M, Pokraev S and van Sinderen M. COSMO: a conceptual framework for service modelling and refinement. In: *Information Systems Frontiers*, 9 (2-3), 2007, pp. 225-244.
- [23] Quartel D and van Sinderen M. On interoperability and conformance assessment in service composition. In: *Proceedings of the Eleventh IEEE International EDOC Enterprise Computing Conference (EDOC 2007)*, 2007, pp. 229-240.
- [24] Quartel D, Dijkman R, van Sinderen M. Methodological support for service-oriented design with ISDL. *Proceedings of the 2nd International Conference on Service Oriented Computing*, 2004, pp. 1-10.
- [25] Rao J and Su X. A Survey of Automated Web Service Composition Methods. In: *Semantic Web Services and Web Service Composition*, LNCS 3387, 2005, pp. 43-54.
- [26] Steffen B, Margaria T, Nagel R, Jörges S and Kubczak C. Model-Driven Development with the jABC. In: *Proceedings of Haifa Verification Conference*, LNCS 4383, 2006, pp. 92-108.
- [27] SWS challenge. <http://sws-challenge.org>
- [28] UDEF. <http://www.opengroup.org/udelfinfo/>