m3pl: A Work-FLOWS ontology extension to extract choreography interfaces

Armin Haller and Eyal Oren

Digital Enterprise Research Institute (DERI)
Galway, Ireland
firstname.lastname@deri.org

Abstract. Cross-organisational interoperability is a key issue for success in B2B e-commerce applications. To achieve this interoperability, choreography descriptions are necessary that describe how the business partners can cooperate. In existing approaches, these choreography descriptions are independent of the internal workflows of the partners.
We present a framework for extracting choreography interface descriptions from internal workflow models. Our approach comprises two steps: first we map internal workflow models into a intermediary formal model, then we generate choreography interfaces from it. In this paper we present m3pl, an ontology extension based upon the First Order Ontology for Web Services (FLOWS) [2]. The extensions provide relations to model workflow views and choreography interfaces.

1 Introduction

Organisations offer business functionalities to their customers, and implement these functionalities in their business processes. For years, organisations have used Workflow Management Systems (WfMSs) to describe and execute their business processes [6]. Underlying these WfMS are different workflow languages with many different metamodels. These workflow languages vary in the available modelling constructs and in the semantics of their constructs. To capture these semantics and to allow interoperability of WfMS the Process Specification Language (PSL) [16] was developed. PSL is an ontology that defines workflow concepts and their semantics. Various extensions have been developed (as part of the PSL standard), including the First Order Logic for Web Services (FLOWS) [2] ontology for modelling (compositions of) Web Services.

With the advent of Service Oriented Computing [13] organisations started to expose their business functionality explicitly as reusable and composable services using standardised protocols such as WSDL and SOAP. Web Services abstract the access to the business functionality from the specifics of programming languages. For using these services organisations provide choreography descriptions written in languages such as WS-CDL [11], Abstract BPEL [20] or ebXML CPP [10]. A choreography describes the message exchange patterns employed by a Web Service interface. These patterns describe how consumers should interact

with the Web Service; they can be described from a global (collaboration) viewpoint or from a local (participant) viewpoint. We will use the term *choreography* for the global viewpoint, and *choreography interface* for the participant's viewpoint[1].

A fundamental limitation in current approaches to model choreographies is its independence to the underlying workflow definitions. Although a few recently published work address the correlation between a choreography interface and its underlying workflow, current approaches do not focus on an automated mapping between them. This independence leads to two problems: (1) if any change occurs in the internal workflow model, choreography descriptions have to be manually synchronised with the workflow definition, and (2) it is not possible to automatically verify consistency of internal workflow descriptions and external choreography interfaces.

This paper presents a framework for combining internal workflow definitions and external choreography descriptions; an overview is shown in Fig. 1. With the framework one can semi-automatically generate choreography interfaces in various languages from workflow models in various languages. The framework is based on PSL [16], an ontology for capturing business processes and FLOWS [2], an extension to PSL for Web Service interactions.

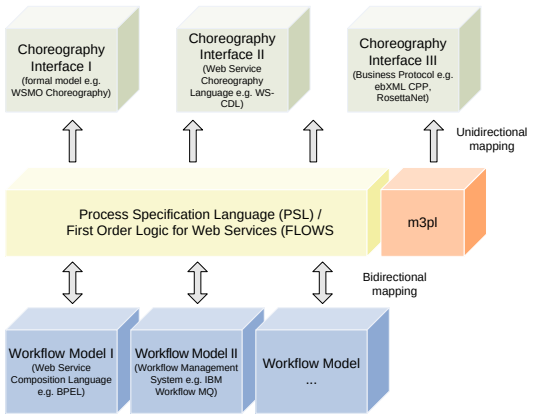

Fig. 1. Relating workflow models to choreography interfaces

The paper is structured as follows: based on a motivating RosettaNet collaboration example described in section 2, we analyse the requirements for our framework in section 3. We present our ontology in section 4. In section 5 we outline the methodology to follow to map from the internal workflow model to m3pl and to extract different choreography interfaces. Finally we discuss related work in section 6 and conclude in section 7.

[1] The choreography interface is also called behavioural interface by Dijkman and Dumas [5] or abstract process in BPEL4WS [20].

2 Motivating Example

In this section we present an example cross-organisational collaboration. We will illustrate the problems that companies face when designing collaborative business processes with a request-for-quote (RFQ) process.

2.1 Current situation

An automotive parts vendor implements and executes his internal processes with IBM Websphere MQ Workflow[2]. One of the vendor's processes concerns the processing of requests for quotes. Figure 2 shows a simplified view of this modelled in MQ Workflow. The symbols on the left of the picture denote a source and sink node and represent the start and end of the MQ Workflow process model. Dashed arrows show data transferred between activities and solid arrows denote the control flow.

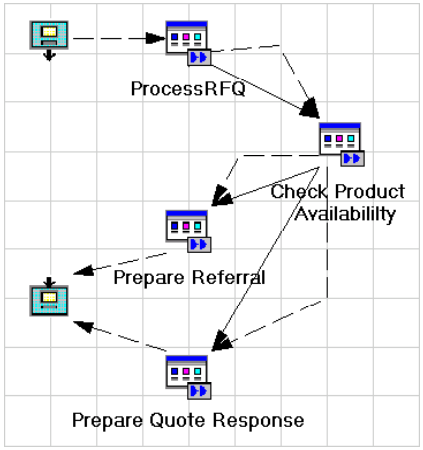

Fig. 2. IBM MQ Workflow RFQ

The process starts with an RFQ from a customer. The vendor checks whether the requested part, say an electric generator, is available in stock and can be delivered within the time specified. If the product is available the vendor prepares a quote, otherwise he returns a referral including the reason for non-delivery.

2.2 Preferred situation

The vendor wants to automate the collaboration with his partners. This would minimise the manual labour by enforcing partners to directly invoke interfaces to

[2] for our analysis we have used v3.4 of the product.

its internal WfMS. An example for such an automation is the initial data input. Currently this data is manually entered into the system; the goal of the vendor is for this input to come directly from the external business partner. To enable automatic collaboration the vendor needs to describe the public view on his business processes. To comply with industry standards this public process should conform to the standardised RosettaNet choreography interface PIP 3A1[3]; which describes a request for quotation.

Figure 3 shows a RosettaNet collaboration and the internal process model described above in a UML activity diagram. Public activities (the RosettaNet PIP 3A1) are displayed in black and private activities in white. The seller's choreography is formed by the black activities in the right swimlane and the buyer's choreography by the black activities in the left swimlane respectively.

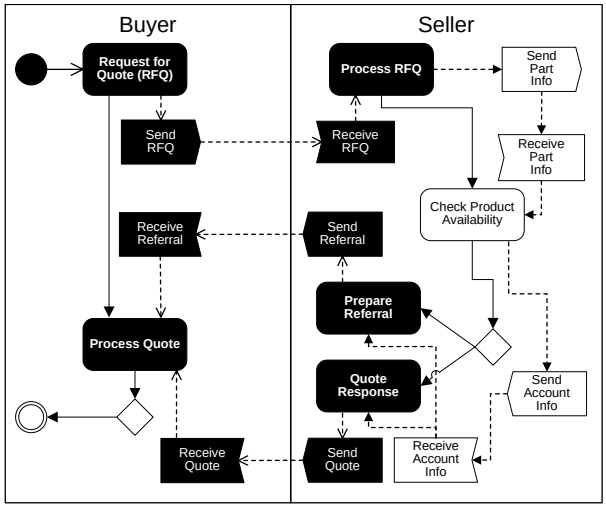

Fig. 3. External Process (RosettaNet PIP)

In this example the internal workflow is straightforward and for the purpose of simplification it is already aligned to an external standard process in terms of a RosettaNet PIP 3A1. Thus it is not difficult to model the external part of the process in any choreography description language. However, in reality the processes can be significantly more complex, and automatic extraction of choreography interfaces is desired.

In order to automatically extract the choreography interface, the internal business process has to be extended by information specific to external processes identified in the following section. Subsequently the model should be extracted to a choreography descriptions language. These features are currently not offered by MQ Workflow or any other WfMS.

[3] `http://www.rosettanet.org/PIP3A1`.

3 Requirements Analysis

We can derive four basic requirements for the above collaboration scenario. They also reflect requirements on a choreography language identified in [1].

1. **Model internal workflow:** we need to model the internal workflow (shown in figure 2) of the business partner whose choreography interface we want to generate (the supplier in this example).
   The internal workflow has to be formally specified to describe the business processes unambiguously. A mere syntactic model could lead to inconsistent interpretations; e.g. a *split* can have different meanings in different models.
2. **Model choreography-related concepts in the workflow:** to generate the choreography interface from the internal workflow we need to add additional annotations. These annotations (such as visibility of activities or role of partners) are not part of the internal workflow, because they are of no significance for workflow enactment, but only for a cross-organisational choreography. It is necessary to annotate:
   (a) the **choreography interface** as a partial view on the internal workflow of the service provider. Choreography interfaces are currently modelled in a multitude of languages. These languages are either **task-flow based** (i.e. WS-CDL, BPEL4WS) or **dependency based** (i.e. WSMO Choreography, OWL-S Process Model). Thus the choreography interface model has to be capable of capturing both modeling alternatives.
   (b) the **visibility** of tasks: some tasks in a collaboration are **private**, other are **public**. Also, some tasks might be publicly visible to one participant, but private to another. The generated choreography interface for one partner should only include the tasks marked as visible to him.
   (c) the **role** a party can play when engaging in collaborations. A role defines the observable behaviour a party exhibits in order to collaborate with other parties. A "buyer" role for example is associated with the purchase of goods or services and the "seller" role is associated with providing those goods or services.
   (d) the **direction of the communication** represents the communication route in a specific interaction and represents constraints on what roles have to be adopted by the participants. A wholesaler for example might play the "seller" role in one interaction and the "buyer" role in some other interaction. A direction relation requires a sending and a receiving **participant**.
   (e) **messages**. As it can be seen in figure 3 messages are used to transfer data between activities. The explicit representation of messages is commonly not part of workflow models. Even if this fundamental approach to model data flow is possible in the underlying workflow model, it is only used to transfer data internally between activities. In the case of a collaboration these messages are sent between the participants and have usually a message type and some payload associated with it.

(f) the **transactional boundaries** of activities to facilitate recovery in the event of a participant failure. The model should allow to define transactional blocks that contain one or many activities that are followed when the effects of a service need to be reversed.

3. **Construct choreography interface from internal workflow:** this is the requirement that drives the framework: the internal workflow model of a particular business process should be reused when constructing a choreography interface, and this process should be automated. Automation requires that mediators are available to different choreography specification representations.
4. **Validate compatibility of choreography interface to internal workflow:** there are several cases where a pre-existing choreography interface has to be validated against an existing workflow model. For example, when partners use a standardised choreography, and extract the choreography interfaces of the participants from this agreement. But a participant might very well already have a workflow model implemented for his business functionality. It is then necessary to verify whether the existent workflow model is compatible to the choreography interface (behavioural equivalence).

4 Ontology for Choreography Interfaces

In what follows we describe the relations in m3pl capturing the requirements identified in the previous section. Our ontology is an extension to PSL [16] modelled in a first-order language. Due to space limitations we do not include the axioms constraining the relations described below. However, where possible we explain how a relation is constrained by the primitive lexical relations axiomatised in PSL-Core.

4.1 Introducing m3pl

To **model the internal workflow** we base our model on PSL [16]. PSL follows a layered approach in the language design, which gives us the resources to express information involving concepts that are not part of the PSL-Core. Thus we can represent arbitrary any workflow model in PSL by introducing extensions which are either defined by relations in the PSL-Core or by axioms that are constraining the interpretation of each new language construct.

The first requirement on the relations associated with the **choreography model** in m3pl is to encompass the two prominent modelling primitives. First we have to be able to extract to different task-flow based choreography description languages, i.e. to Abstract BPEL [20], WS-CDL [11] and ebXML CPP [10] and second to dependency-based ontology-based choreography descriptions, i.e. WSMO Choreography [18], OWL-S Process Model [12]. PSL provides relations to incorporate both workflow modelling primitives.

The m3pl extension offers a model to describe the choreography interface of some internal workflow model, whereas the choreography interface represents a

model of some functionality (i.e. services). The functional entity in m3pl is a member of the set of such services in the universe of discourse of the interpretation. Services are reusable behaviours within the domain and relate to activities in PSL. A service occurrence models an occurrence of a PSL complex activity that is associated with the service.

```
service(?functional_entity)
service_activity(?functional_entity,?activity)
service_occurrence(?functional_entity,?occurence)
```

Listing 1.1. Service Relations

The views extension defines a relation to give one the possibility to restrict the **visibility** of specific activity occurrences to a certain role and thus create different views [4, 17] on a workflow model. The `visible(?occurrence,?role)` relation associates an activity occurrence to a role. By relating a participant to a specific role the visibility of activity occurrences is guaranteed to be constrained to the defined business partner only.

```
visible(?occurrence,?role)
```

Listing 1.2. Visibility Relation

Roles define the conversational relationship between two or more partners by defining the part played by each of them in the conversation. The `partner_link(?role,?functional_entity)` relation models such conversational relationships. The `participate(?agent,?role)` relation is used to relate an organisation to a role that it is playing in a specific collaboration.

```
partner_link(?role,?functional_entity)
participate(?agent,?role)
```

Listing 1.3. Conversational Role Relations

A key element in choreography description languages as identified above is the notion of **messages**. Since there exist different strategies of data passing in commercial workflow systems and workflow models, we offer relations which allow to model all three strategies as identified in [14].

Data is modelled with predicates and terms in first-order language. They act as fluents whose values may change as the result of service occurrences. Similar to FLOWS we use the `described_by` relation to associate a message_type to a fluent. Multiple fluents might be associated with one message_type, which should be interpreted as a conjunction of them.

Further we allow to associate the fluent to a channel. The `send` and `receive` relations are used to "transfer" fluents from one activity occurrence to the next. Both relations are associated with the `participates_in` relation of PSL, which is used to constrain which objects are involved in a particular occurrence of an activity. Thus in this data passing modelling approach no occurrence of an activity can begin without first receiving, and cannot send before it ends. The read relation is similar to receive, but with a weaker ontological commitment on the occurrence. It is not required that a send occurrence preceded the occurrence associated with the read relation.

```
described_by(?message_type, ?fluent)
send(?fluent, ?channel, ?occurrence)
receive(?fluent, ?channel, ?occurrence)
read(?fluent, ?occurrence)
input_port(?channel,?occurrence)
output_port(?channel,?occurrence)
```

Listing 1.4. Data Modelling Relations

Channels are used to model message-based communication as used in Web Services. We adopt the relations offered in FLOWS [2]. However, we do not relate channels to service occurrences, but to activity occurrences. Channels are a way to model explicit data passing, but are not required to exist since fluents can be related to activity occurrences via the read relation.

In order to capture **dependency based** workflow models every atomic activity occurrence can be associated with preconditions and effects. The occurrence of an atomic activity therefore transforms an initial state of the world (preconditions) into a final state that represents the world (effects) after the execution. Essentially the two relation are similar to the send and receive relation described earlier, since preconditions and effects are also represented by fluents in the ontology. The only difference being that they are not associated with a channel. However, they are a different concept in the real world, since preconditions and effects are not necessarily constraints on data, but might be constraints on the existence of objects.

```
precondition(?conditional_fluent,?occurrence)
effect(?conditional_fluent,?occurrence)
```

Listing 1.5. Dependency relations

All **task-flow based** choreography languages use control constraints to model the control flow of Web Services. However they are not natively included in PSL. Thus we reuse the definitions in FLOWS with minor extensions, which are together sufficient to model the majority of constructs available in choreography description languages.

```
sequence(activity)
split(activity)
IfThenElse(activity)
LoopUntil(activity)
wait(activity)
```

Listing 1.6. Control Constraint Relations

A `sequence` relation specifies that all subactivity occurrences of a complex activity are totally ordered. It corresponds to a `soo_precedes` relation in the Duration and Ordering Theory of PSL.

The subactivity occurrences of a split (corresponds to a flow construct in BPEL) activity are constrained by two relations from the PSL ontology. One subactivity occurrence `soo_precedes` any number of subactivity occurrences while they are `strong_parallel` to each other.

The `IfThenElse` activity is a nondeterministic activity such that the subactivity occurrences are constrained on the state conditions that hold prior to

the activity occurrence. The use of `IfThenElse` is equivalent to a conditional activity in PSL.

The subactivity occurrences of a `LoopUntil` activity occur multiple times until the state condition is satisfied. It is equivalent to a conditional activity in the PSL ontology whose occurrences are repetitive, whereas the occurrence trees have different structure, depending on the cardinality.

The subactivity occurrences of a wait activity delays the process for a certain timeperiod or until a timepoint has passed.

Error handling in collaborative interactions is as important as transactional support in local application environments. The use of ACID transactions [7] is not feasible in collaborations, because locks on some activities cannot be maintained for periods of an asynchronous interaction. Error handling therefore relies heavily on the well-known concept of compensation. That is, if some state occurs alternate activities are performed which reverse the effects of a previous activity that was carried out and caused the error. To model such situations, we add failure handling activities, which are conditioned over an exception state raised by an earlier activity occurrence.

5 A methodology to extract choreographies

In the following section we show the applicability of the m3pl ontology extensions by outlining the methodology to follow when extracting choreography descriptions from internal workflow definitions. We apply the methodology to our example introduced in section 2, whereas we will focus on the supplier.

1. First the syntactic model (c.f. figure 2) underlying most Workflow Management Systems has to be lifted to the PSL/FLOWS ontology. In order to generate it automatically, mapping rules are required. This is not a trivial task since the generic mapping rules have to capture the operational semantics of the underlying WfMS.
   In our scenario the supplier models and enacts its business processes with IBM MQ Workflow. The workflow model is serialised in a proprietary description file called *.ftl*. We have identified the mapping rules necessary to translate our example workflow. However, it is not in the scope of this paper to define a generic mapping framework for arbitrary any workflow in IBM MQ Workflow. Listing 2.1. shows a snippet of the model from figure 2, i.e. the *Check Product Availability* activity followed by a decision point and either the *Prepare Referral* or the *Prepare Quote Response* activity. The full listing can be found at `http://m3pe.org/ontologies/PSLRFQ.kif`.

$state(productListedOK)$
$state(productListedFailed)$

$\forall(?occRFQWorkflow)$
$occurence\_of(?occRFQWorkflow,\ RFQWorkflow)$

$\Rightarrow \exists(?occProcessRFQ,\ ?occCheckProductAvailability)$
$(occurrence\_of(?occProcessRFQ,\ ProcessRFQ) \wedge$
$occurrence\_of(?occCheckProductAvailability,\ CheckProductAvailability) \wedge$
$subactivity\_occurrence(?occProcessRFQ,\ ?occRFQWorkflow) \wedge$
$subactivity\_occurrence(?occCheckProductAvailability,\ ?occRFQWorkflow) \wedge$
$root\_occ(?occProcessRFQ) \wedge$
$soo\_precedes(?occProcessRFQ,\ ?occCheckProductAvailability,\ ?occRFQWorkflow)) \wedge$

$(holds(productListedFailed,\ ?occCheckProductAvailability) \wedge$
$not(productListedOK,\ ?occCheckProductAvailability))$
$\Rightarrow \exists(?occPrepareReferral)$
$(occurrence\_of(?occPrepareReferral,\ PrepareReferral) \wedge$
$subactivity\_occurrence(?occPrepareReferral,\ ?occRFQWorkflow) \wedge$
$leaf\_occ(?occPrepareReferral,\ ?occRFQWorkflow)) \wedge$

$(holds(productListedOK,\ ?occCheckProductAvailability) \wedge$
$not(productListedFailed,\ ?occCheckProductAvailability))$
$\Rightarrow \exists(?occPrepareQuoteResponse)$
$(occurrence\_of(?occPrepareQuoteResponse,\ PrepareQuoteResponse) \wedge$
$subactivity\_occurrence(?occPrepareQuoteResponse,\ ?occRFQWorkflow) \wedge$
$leaf\_occ(?occPrepareQuoteResponse,\ ?occRFQWorkflow)) \wedge$

Listing 2.1. Snippet of internal workflow in PSL/FLOWS

2. Next, the ontology instance representing a semantically equivalent model to the underlying workflow definition has to be annotated with choreography specific constructs from m3pl. Since domain experts knowledgeable of what parts of the workflow model are required to be published to partners and technology experts competent in defining the message exchange are not necessarily familiar with formal frameworks (i.e. First Order Logic), editor support is required to ease the annotation task. We are currently building a domain specific GUI-based tool for annotating the extracted model with concepts defined in our ontology.
   However, in the context of this paper we have manually annotated the generated ontology instance without tool support. The complete annotated model can be found at `http://m3pe.org/ontologies/RFQm3pl.kif`. This annotation is comprised of relations defined in section 4 capturing the collaborative role model, the visibility constraints, the message descriptions and its passing directions.
   Listing 2.2. shows the m3pl annotations added to the snippet of our internal workflow from Listing 2.1.

$service(RFQProcessing)$
$service\_activity(RFQProcessing,\ RFQWorkflow)$
$service\_occurrence(service,\ activity\_occurrence)$
$role(Customer,\ RFQProcessing)$
$role(Supplier,\ RFQProcessing)$
$participant(Bosch,\ Supplier)$
...
$\forall(?occRFQWorkflow)$
$occurence\_of(?occRFQWorkflow,\ occRFQWorkflow)$

$\Rightarrow \exists(?occProcessRFQ,\ ?occCheckProductAvailability)$
...
$visibility(?occProcessRFQ,\ Customer) \wedge$
$visibility(?occCheckProductAvailability,\ Supplier) \wedge$
$input\_port(transmitRFQ,\ ?occProcessRFQ) \wedge$
...
$\Rightarrow \exists(?occPrepareReferral)$
...
$visibility(?occPrepareReferral,\ Customer) \wedge$
$output\_port(transmitReferral,\ ?occPrepareReferral)) \wedge$
...
$\Rightarrow \exists(?occPrepareQuoteResponse)$
...
$visibility(?occPrepareQuoteResponse,\ Customer) \wedge$
$output\_port(transmitQuoteResponse,\ ?occPrepareQuoteResponse) \wedge$

Listing 2.2. Annotation added to the extracted PSL/FLOWS model

3. Based on this ontological model choreography interfaces for each partner in the collaboration can be generated. Most importantly all activities marked in the previous step as private to the supplier will be omitted in the choreography interface. The split modelled in the choreography interface published to the customer is abstracted in a way that the evaluation of the condition is non-deterministic to the buyer and is modelled in the choreography interface as follows: $(holds(True,\ ?occCheckProductAvailability) \wedge not(False,\ ?occCheckProductAvailability))$
4. If required a multiparty choreography can be assembled. Since our model is based on a formal ontology this matching process can be on different levels of abstraction. The simplest matching algorithm compares the message type and the counterparting message passing direction. More complex matching can include full reasoning over the first-order models proving the equivalence of two choreography interface models.
5. Finally, choreography descriptions in existing languages such as WS-CDL, Abstract BPEL or ebXML CPP can be generated. Similar to step one, mapping rules have to be defined for each choreography description language. Different to above though the mapping is unidirectional, since the choreography descriptions represent an abstraction omitting information necessary in the internal workflow definition

An example extracted choreography interface from our model in a BPEL description is shown in listing 2.3. The interface starts with the definition of the partners, generated from the manually added annotations. The actual process starts at line 8 and contains the three workflow activities as *invoke* and *receive* operations. The *check-product-availability* activity, the split conditions and the internal data transfer are omitted from the choreography interface since they were marked as private information.

```
1  <wsdl>
2   <plnk:partnerLinkType name="buyerSellerRelation">
3    <plnk:role name="seller"><plnk:portType name="rfqpw"/></plnk:role>
4    <plnk:role name="buyer"><plnk:portType name="rfqpwCallback"/></plnk:role>
5   </plnk:partnerLinkType>
6  </wsdl>
7
8  <process name="RFQProcessing">
9   <partnerLink name="buyerSellerRelation" partnerLinkType="lns:buyerSellerRelation"
10    myRole="seller" partnerRole="buyer"/></partnerLinks>
11  <variables>
12   <variable name="rfqMessage" messageType="lns:rfq"/>
13   <variable name="quoteMessage" messageType="lns:quote"/>
14   <variable name="referralMessage" messageType="lns:referral"/>
15  </variables>
16  <sequence name="main">
17   <receive name="processRFQ" partnerLink="buyerSellerRelation"
18     portType="lns:rfqpw" operation="initiate" variable="rfqMessage"/>
19   <assign><copy>
20    <from opaque="yes"/><to variable="condition" property="xsd:boolean"/>
21   </copy></assign>
22   <switch name="quoteDecision">
23    <case condition="if bpws:getVariableData('condition') = true">
24     <invoke name="prepareReferral" partnerLink="buyerSellerRelation"
25      portType="lns:rfqpwCallback" operation="onResult"
26      inputVariable="quoteMessage"/>
27    </case>
28    <otherwise>
29     <invoke name="processQuote" partnerLink="buyerSellerRelation"
30      portType="lns:rfqpwCallback" operation="onResult"
31      inputVariable="referralMessage"/>
32    </otherwise>
33   </switch>
34  </sequence>
35 </process>
```

Listing 2.3. Abstract BPEL description

6 Related Work

Our work is most closely related to several approaches to views on process models, i.e. [3, 4, 17, 15, 21].

Chebbi et al. [3] propose a view model based on Petri Nets. They introduce cooperative activities, which can be partially visible for different partners. The approach is validated on mappings from two different WfMSs. However, the model requires n2 mappings and does not explain how to model the data aspect, i.e. the message transfer between partners.

Chiu et al. [4] present a cross-organisational meta model which is implemented in XML. Similar to the cooperative activities in [3] the model provides so called cross-organisational communications, which allow to define message transfer and its direction. The model is implemented in an extension to the ADOME-WfMS, called E-ADOME. The model deals only with sequential activities in the abstracted view and does not tackle an integrated approach in choreography extraction and requires the specific model to be used in the E-ADOME tool.

Schulz and Orlowska [17] introduce a Petri-Net based state transition approach that binds states of private workflow tasks to their adjacent workflow view-task, where existing workflows are augmented by means of one or more activities or sub-workflows of an external workflow. The model is conceptualised in a supporting architecture. The approach identifies mappings in its conceptual architecture, but it does not describe how to integrate different workflow models. Further the approach abstracts from the data aspect.

Sayal et al. [15] introduce service activities (that represent trade partner interaction) as workflow primitives, but their approach is specific to one workflow modelling tool and addresses neither workflow integration nor choreography interface extraction.

Zhao et al. [21] define a relative workflow model representing the view of one partner on local workflows of another partner. They present composition rules how to generate the relative workflow and a simple matching algorithm to connect two local workflow process. Similar to the other approaches it is meta model independent.

Several approaches address interoperability issues between Workflow Management Systems (WfMS), such as Mobile [9], Meteor [19] and CrossFlow [8]. However, all of these approaches require a pre-established partner agreement on the semantics of the process models. Further they were all proposed before the advent of Service Oriented Architectures and therefore do no deal with standard choreography description languages.

7 Conclusion

In existing approaches, choreography descriptions are independent of the internal workflows of the partners and have to be manually mapped.

We presented m3pl, an ontology extension to PSL and FLOWS to formally capture choreography-specific information. The ontology extension together with PSL can act as a connecting ontology to integrate different workflow models and subsequently extract external process models.

We have shown how the framework can be used to extract a choreography interface of an example workflow in a RosettaNet collaboration. This initial validation is based on the translation of an example workflow represented in IBM Websphere MQ Workflow to PSL, which is then manually annotated with relations offered in the m3pl extension to further extract a BPEL process.

One direction of our future work is to check the equivalence of to choreography models. Given a choreography interfaces it is desirable to verify whether it is compatible with the choreography interface of a partner and -if they are indeed compatible- to construct a multiparty choreography.

Acknowledgment

This material is based upon works supported by the Science Foundation Ireland under Grant No. SFI/04/BR/CS0694.

References

1. D. Austin, A. Barbir, E. Peters, and S. Ross-Talbot. Web Services Choreography Requirements. Working draft, W3C, Mar. 2004.
2. S. Battle, *et al.* Semantic Web Services Ontology (SWSO). Member submission, W3C, Sep. 2005.
3. I. Chebbi, S. Dustdar, and S. Tata. The view-based approach to dynamic inter-organizational workflow cooperation. *Data Knowl. Eng.*, 56(2):139–173, 2006.
4. D. K. W. Chiu, *et al.* Workflow view driven cross-organizational interoperability in a web service environment. *Inf. Tech. and Management*, 5(3-4):221–250, 2004.
5. R. Dijkman and M. Dumas. Service-oriented design: A multi-viewpoint approach. *Int. Journal of Cooperative Information Systems*, 13(4):337–368, Dec. 2004.
6. D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.
7. J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Francisco, 1993.
8. P. Grefen, K. Aberer, H. Ludwig, and Y. Hoffner. CrossFlow: Cross-organizational workflow management for service outsourcing in dynamic virtual enterprises. *IEEE Data Engineering Bulletin*, 24(1):52–57, 2001.
9. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation*. Int. Thomson Computer Press, 1996.
10. N. Kartha *et al.* Collaboration-protocol profile and agreement specification, v2.1, Apr. 2005.
11. N. Kavantzas *et al.* Web services choreography description language, Nov. 2005.
12. D. Martin, *et al.* Owl-s: Semantic markup for web services. Member submission, W3C, 2004. Available from: `http://www.w3.org/Submission/OWL-S/`.
13. M. P. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Communications of the ACM*, 46(10):25–28, 2003.
14. N. Russell, A. H. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Workflow data patterns. FIT-TR-2004-01, Queensland University of Technology, 2004.
15. M. Sayal, F. Casati, U. Dayal, and S. Ming-Chien. Integrating workflow management systems with business-to-business interaction standards. In *Proc. of the 18th Int. Conf. on Data Engineering*, pp. 287–296. 2002.
16. C. Schlenoff, *et al.* Process specification language (PSL): Overview and version 1.0 specification. Tech. Rep. NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD, 2000.
17. K. A. Schulz and M. E. Orlowska. Facilitating cross-organisational workflows with a workflow view approach. *Data Knowl. Eng.*, 51(1):109–147, 2004.
18. J. Scicluna, A. Polleres, and D. Roman. Ontology-based choreography and orchestration of wsmo services. Wsmo working draft v0.2, DERI, 2005. Available from: `http://www.wsmo.org/TR/d14/v0.2/`.
19. A. Sheth, *et al.* The METEOR workflow management system and its use in prototyping significant healthcare applications. In *Proc. of the Toward an Electronic Patient Record Conf. (TEPR'97)*, pp. 267–278. Nashville, TN, USA, 1997.
20. S. Thatte *et al.* Business process execution language for web services, v1.1, May 2003.
21. X. Zhao, C. Liu, and Y. Yang. An organisational perspective on collaborative business processes. In *Proc. of 3rd Intl. Conf. on Business Process Management*, pp. 17–31. Sep. 2005.