From workflow models to executable Web service interfaces

Armin Haller, Mateusz Marmolowski
Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway
firstname.lastname@deri.org

Walid Gaaloul
TELECOM & Management SudParis (ex INT)
UMR SAMOVAR
walid.gaaloul@it-sudparis.eu

Eyal Oren
AI Department
Vrije Universiteit Amsterdam
eyal@cs.vu.nl

Brahmananda Sapkota, Manfred Hauswirth
Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway
firstname.lastname@deri.org

Abstract

Workflow models have been used and refined for years to execute processes within organisations. To deal with collaborative processes (choreographies) these internal workflow models have to be aligned with the external behaviour advertised through Web service interfaces. However, traditional workflow management systems (WfMS) do not offer this functionality. Simply sharing and merging process models is often not possible, because workflow management lacks a widely accepted standard theory for workflow models. Multiple research and standardisation efforts to integrate different workflow theories have been proposed over the years. XPDL is the most widely used standard for process model interchange and supported by over 80 systems. However, XPDL also lacks the possibility to relate a workflow model to its possible choreography interface abstractions. To remedy this situation, we propose to abstract the XPDL model to a higher-level model, perform the integration and the compaction algorithms at that level and then ground it back to the desired choreography models. We develop and use an integrated ontology which is based on the XPDL standard for this purpose. To facilitate the abstraction and grounding, we present a mapping procedure to automatically translate XPDL and BPMN workflow models into this ontology. After translation, these models are annotated with a parameterised role model and other collaborative properties. We present a compaction procedure that automatically maps the annotated models into external choreography interfaces that expose only the relevant information for a particular partner collaboration. Our procedure is agnostic with respect to the target choreography model. We demonstrate our approach using WSMO choreographies which enables us to automatically generate interface models from any WfMSs that supports XPDL export.

1 Introduction

Internet technologies enable organisations to connect their supply chains more tightly to form virtual enterprises [16] where resources are shared to improve the efficiency of their cooperation. Service-Oriented Computing [15] promises to facilitate the sharing of resources by a network of cooperating services, loosely coupled into flexible business processes that span multiple organisations. The composition of services ensures the correct cooperation between business partners according to a pre-defined business logic. This business logic is described in *choreography models*, stating conversational patterns with which the services can be accessed. A *choreography interface*, also known as "interface behaviour" [7], is a subset of a choreography model that describes the behaviour of one participant in a collaboration. Conceptually, a choreography interface of one partner can be regarded as an abstracted view on a workflow model [5, 18, 7, 4]. However, traditional workflow management systems (WfMS), Web service compositions languages such as BPEL [1] and choreography models such as WS-CDL [13] lack a connection between the workflow model and the choreography model. This disconnection leads to two problems:

- choreography interfaces need to be created and synchronised with the workflow model manually, and
- the consistency of choreography interfaces to the workflow model is not guaranteed.

An obstacle in connecting workflow models with choreography models is the variety of existing workflow languages, workflow metamodels, and choreography languages. Multiple research and standardisation efforts to integrate different workflow theories and models have been proposed over the years. The XML Process Definition Lan-

guage (XPDL) is the most widely used standard for *process model interchange* and supported by over 80 systems, among them the market leaders in WfMSs. XPDL, since version 2.0, also serves as serialisation of BPMN[1] models, a standard graphical representation for workflow models. However, XPDL also lacks the possibility to relate the workflow model to its possible choreography interface abstractions. Due to its weak data model it is further challenging to integrate different business documents whose exchange among workflows represents the foundation of any collaborative process. To address this challenge we extend the multi-meta model process ontology (*m3po*) introduced in [10] with concepts for a full formalisation of the metamodel of XPDL. The *m3po* ontology presented in this paper explicitly models the complete semantics of XPDL. The integrated *m3po* ontology is used as shared representation to perform the integration and the compaction algorithms at that level and then ground it back to the desired choreography models. The advantage of this approach besides others is that we can use a Web ontology language for the formalisation of our model to link data with established business document standards.

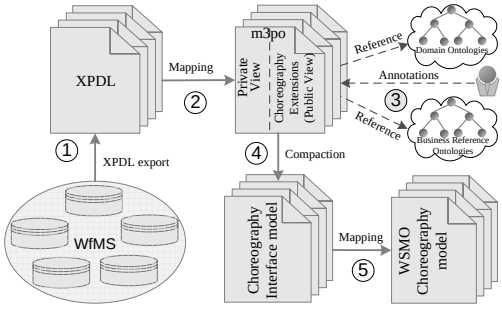


Figure 1: Overview: from workflow to choreography

Our approach starts from the XPDL output of a workflow management system (step 1 in Fig. 1). Our system implements mapping rules for the complete XPDL standard. The translation from an XPDL document (step 2) results in ontological instances according to the *m3po* metamodel. These instances are annotated by a business expert with collaboration-specific information (step 3), using an ontology editor such as WSMO Studio[2]. These annotations are the only required manual input to our mapping process, but they are lightweight and mainly define the visibility of workflow tasks to a specific partner in the collaboration. Based on the annotated workflow model our compaction rules (step 4) extract a choreography interface model for a specific partner. The resulting abstraction can be mapped (step 5) to an executable choreography interface model. Each step is described in the subsequent sections.

[1]See `http://www.bpmn.org/`

[2]See `http://www.wsmostudio.org/`

1.1 Motivating Example

In our example, we assume that the collaborating partners agree on using a standard choreography such as the one defined by RosettaNet. RosettaNet[3] defines inter-organisational processes within Partner Interface Processes (PIPs). RosettaNet PIPs are mainly used by companies in the ICT domain to control their external collaborations. Our motivating example describes an internal process (workflow) of a *Supplier*, and the *desired* public collaboration in a "Request for Quote" (RFQ) process with two partners, a *Customer* and a *Transaction Bank*. The public process with the *Customer* follows the standard process PIP3A1 defined by RosettaNet. Fig. 2 depicts a BPMN diagram of the global conversation (choreography) between the *Supplier* and the *Customer*. Private activities in the workflow of the *Supplier* are depicted as black boxes, whereas white activity boxes represent public activities in the collaboration with the *Customer* and grey activities with the *Transaction Bank*. The *Supplier*'s provided interface to a *Customer* is formed by the white activities in its pool. Similarly, the complementary *Customer's* requested interface is provided by the white activities in its pool.

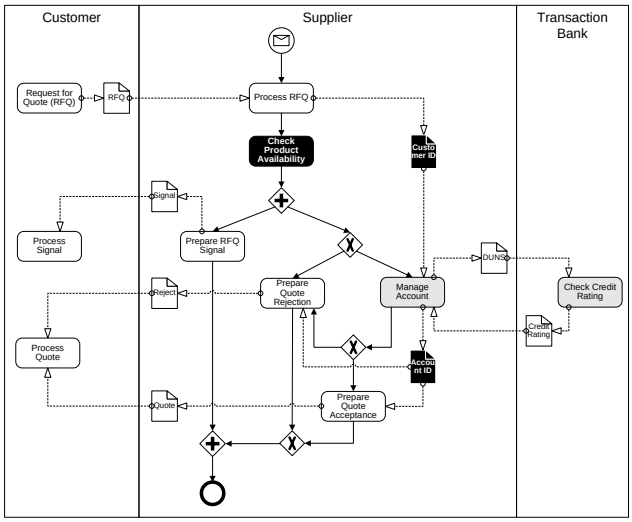
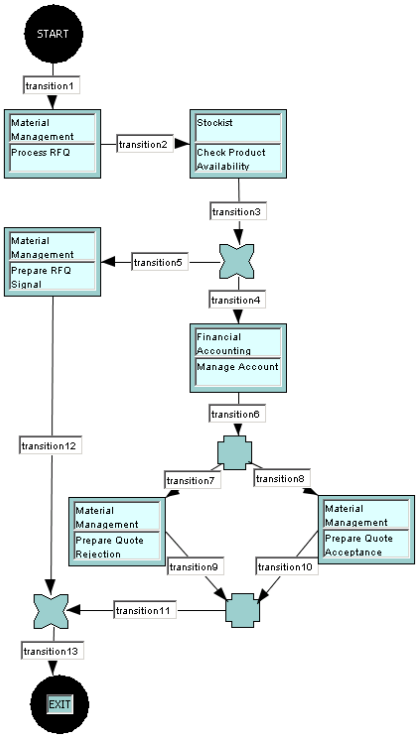


Figure 2: RFQ process including external communication

Fig. 2 shows only the behavioural aspect and parts of the informational aspect. Organisational and operational aspects of the model such as the role assignment of the "Check Product Availability" activity are not shown for the sake of clarity of the model. However, parts of the organisational aspect can be seen in Fig. 3 (cf. role assignments such as the "Stockist" in the "Check Product Availability" activity).

We assume that the *Supplier* implements and executes his internal workflow in a traditional WfMS. We have chosen Fujitsu iFlow (now Fujitsu Interstage Suite) as an example WfMS. The rationale for this choice is that Fujitsu iFlow

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




```
<?xml version="1.0"?>
<Package xmlns:xpdl="http://www.wfmc.org/2002/XPDL1.0" xmlns:xsi="
        http://www.w3.org/2001/XMLSchema−instance" Id="1551"
        Name="RFQ process">
    <PackageHeader>
        <XPDLVersion>1.0</XPDLVersion>
        <Vendor>Fujitsu Software Corporation</Vendor>
        <Created>Thu Dec 04 17:16:30 GMT 2008</Created>
    </PackageHeader>
    <WorkflowProcesses>
        <WorkflowProcess Id="1551" Name="Thesis example process"
                AccessLevel="PUBLIC">
            ...
            <Participants>
                <Participant Id="Material Management" Name="Material
                        Management">
                    <ParticipantType Type="ROLE"/>
                </Participant>
                <Participant Id="Financial Accounting" Name="Financial
                        Accounting">
                    <ParticipantType Type="ROLE"/>
                </Participant>
                <Participant Id="Stockist" Name="Stockist">
                    <ParticipantType Type="ROLE"/>
                </Participant>
            </Participants>
            <Activities>
            ...
                <Activity Id="1558" Name="Process RFQ">
                    <Performer>Material Management</Performer>
                    <StartMode><Automatic/></StartMode>
                    <FinishMode><Automatic/></FinishMode>
                    <TransitionRestrictions>
                        <TransitionRestriction>
                            <Join Type="XOR"/>
                            <Split Type="XOR">
                                <TransitionRefs>
                                    <TransitionRef Id="1563"/>
                                </TransitionRefs>
                            </Split>
                        </TransitionRestriction>
                    </TransitionRestrictions>
                    ...
                </Activity>
                ...
        </WorkflowProcess>
    </WorkflowProcesses>
</Package>
```

```
instance PackageType1551 memberOf PackageType
  hasName hasValue _string("RFQ process")
  hasPackageHeader hasValue PackageHeader_1
  hasProcessType hasValue ProcessType1551

instance PackageHeader_1 memberOf PackageHeader
  hasXPDLVersion hasValue _string("1.0")
  hasVendor hasValue _string("Fujitsu Software Corporation")
  hasCreated hasValue _string("Thu Dec 04 17:16:30 GMT 2008")

instance ProcessType1551 memberOf ProcessType
  hasName hasValue _string("Thesis example process")
  hasAccessLevel hasValue PUBLIC
  hasParticipant hasValue MaterialManagement
  hasParticipant hasValue FinancialAccounting
  hasParticipant hasValue Stockist
  hasActivity hasValue Activity1558
  hasTransition hasValue Transition1563

instance MaterialManagement memberOf Participant
  hasName hasValue _string("Material Management")
  hasParticipantType hasValue ParticipantType_2

instance Financial Accounting memberOf Participant
  hasName hasValue _string("Financial Accounting")
  hasParticipantType hasValue ParticipantType_2

instance Stockist memberOf Participant
  hasName hasValue _string("Stockist")
  hasParticipantType hasValue ParticipantType_2

instance Activity1558 memberOf Activity
  hasName hasValue _string("Process RFQ")
  hasPerformer hasValue MaterialManagement
  hasTransitionRestriction hasValue TransitionRestriction_2

instance TransitionRestriction_2 memberOf TransitionRestriction
  hasJoin hasValue Join_1
  hasSplit hasValue Split_2

instance Transition1563 memberOf Transition
  hasFrom hasValue Activity1558
  hasTo hasValue Activity1557

instance Join_1 memberOf Join
  hasType hasValue _string("XOR")

instance Split_2 memberOf Split
  hasType hasValue _string("XOR")
```

Figure 3: RFQ process in Fujitsu iFlow, the corresponding XPDL output (step 1) and the translated m3po instance (step 2)

uses a proprietary workflow model but offers XPDL import/export facilities. Since many other WfMS use proprietary models as well [22] and about 80 in total offer XPDL export, our choice is representative for traditional WfMSs. Fig. 3 shows the workflow modelled in Fujitsu iFlow. Snippets of the XPDL exported by Fujitsu iFlow are shown in the middle part of Fig. 3.

Although the *Supplier* executes the internal workflow model in a WfMS, the communication with a *Customer* is not managed as such. The automation of the collaboration could minimise the manual labour involved by enforcing partners to directly invoke interfaces to its internal WfMS. All data input from collaboration partners should be through electronic data exchange. One example for such an input is the initial "Request for Quote" message. To enable automatic collaboration the *Supplier* needs to describe the public view on its workflow, possibly adapt it, define the mapping to a collaborative process model and eventually execute the message exchange patterns in the choreography interface. Considering the fragmented market of WfMSs and the lack of a standardised process model, it is likely that different *Customers* and also the *Transaction Bank* will use different WfMSs with different metamodels and languages. Lacking a uniform metamodel, defining and executing collaborative choreographies across different WfMSs is not feasible. We therefore propose an integrated ontology of both internal workflow and choreography aspects that is able to represent multiple metamodels and links data with established business document standards. Using our mapping rules, the example workflow model is translated into our integrated ontology; the result is shown on the right side of Fig. 3.

2 Related Work

Chiu et al. [5] present a metamodel and prototype workflow system that includes cross-organisational communication properties. Schulz & Orlowska [18] build upon the idea of workflow views and introduce a state-transition approach that binds states of private workflow tasks to a corresponding view-task in a Petri-net. In contrast to the work of Chiu et al. [5], they identify mappings between the workflow and the choreography model in the conceptual architecture. However, both approaches are restricted to their own workflow language, do not integrate different workflow models, and ignore the data aspect.

Martens [14] verifies the consistency between executable and abstract BPEL. The verification is based on an ab-

stracted communication graph (choreography interface) using Petri-nets. The approach focuses on determining the behaviour equivalence of abstract processes, rather than extracting choreographies. Bobrik et al. [2] create personalised process views based on parameterised operations, but only the behavioural aspect is considered and the result is meant for visualisation only.

Chebbi et al. [4] extract choreography models from workflows by defining the partial visibility of workflow activities and their resources. They present rules how to come from an internal workflow to a cooperative one in two steps. Their approach is based on Petri-nets and their contraction rules are meant to generate a collaborative choreography model. In contrast, our privacy annotations are similar, but we then compact to a choreography interface model first, followed by the generation of many different execution models (for each partner). Eshuis & Dehnert [8] do construct one parameterised view per consumer and also consider consistency criteria between the public process and internal one. Still, both approaches are limited to the behavioural aspect without catering for e.g. data compatibility.

Tran et al. [21] also build on workflow views, and propose an automated integration of models at different abstraction levels. They present an extensible reverse-engineering tool-chain to automatically populate various view models from existing BPEL process descriptions and then generate executable code from these views. Their approach is closest to ours, with the difference that we start from an interchange model, allowing us to integrate also non-BPEL models and to generate an executable choreography model in the final step.

Roman et al. [17] propose a methodology for generating a WSMO choreography model from graph-based specifications. Based on the WSMO choreography model they propose a faithful extension that is based on Concurrent Transaction Logic (CTR). The work is similar to ours in regard to the translation from a graph-based model to WSMO choreographies and is complimentary in regard to the CTR-extensions to the WSMO model. However, they only propose a translation from graph-based models, but do not cover the choreography extraction issue from workflow models.

3 Translating XPDL to *m3po*

The *m3po* ontology is not meant for direct modelling, but as an interchange format where instances are automatically generated by the conversion of XPDL documents. The actual conversion in our tool[4] involves two steps. First, the *m3po* ontology, described in the next section, is loaded. The ontology definition acts as the language grammar, including the concept hierarchy, its attributes and their cardinalities. Then, the actual XPDL document is analysed and translated into a corresponding *m3po* document

The detailed process is depicted in the UML Activity Diagram in Fig. 4. First, the XML file is parsed and for each node (both text and element) we check whether the corresponding concept or attribute is defined in *m3po* or should be ignored. For this lookup, the names of *element nodes* are mapped using a special dictionary that maps synonyms defined by the WfMC to concepts in *m3po*.

After the name mapping the converter checks whether a concept is actually defined in *m3po* or not. If it is, the process continues with the instance name construction. There are two cases that may occur. Either the ID is explicitly defined in the XML document or the tool generates a unique identifier by a combination of the element name and a hash of the creation date. When a new instance is created all attributes within the concept are parsed, their types are checked and if required they are added to the ontology.

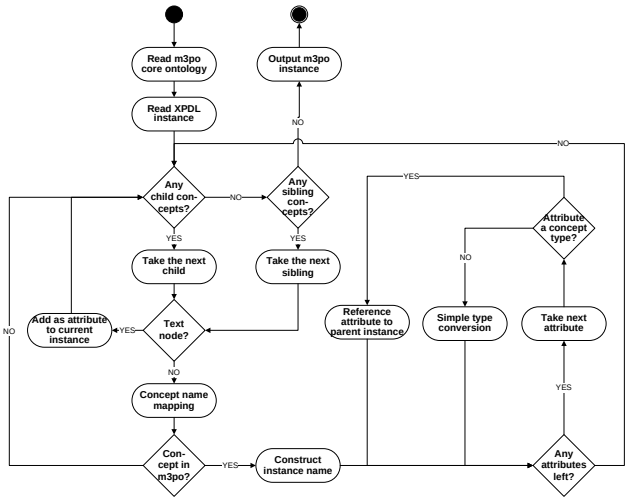


Figure 4: Processing steps of the conversion algorithm

The result of the conversion process is an instantiated *m3po* ontology that represents the knowledge encoded in the XPDL document. We have verified the correctness of the translation tool with respect to the XPDL metamodel. Note that the translation procedure is not hard-coded into the converter tool, but based on declarative mapping rules. This declarative approach makes the converter flexible with respect to changes in either XPDL or *m3po* since new rules can be added to encompass those changes.

4 Modelling BPMN and XPDL in *m3po*

In this section we briefly present the concepts in *m3po* concerned with the modelling of the XPDL and BPMN metamodel. We classify the workflow topics covered in the ontology into a functional, behavioural, informational, or-

[4]See `http://www.m3pe.org/oXPDL/`

ganisational, and operational aspect [12]. We describe the concepts related to each aspect and illustrate each section with a UML class diagram that presents the main concepts and attributes in *m3po*. Please note that, for brevity, each aspect is presented in a separate UML diagram, not showing the relations between concepts in different aspects[5].

The core *m3po* ontology [10] covers all of XPDL/BPMN. The ontology is modelled in WSML [6] and consists of 130 concepts and 50 logical axioms. The translation from XPDL to *m3po* is fully automatic, as described in the previous section. In each section, we also present choreography-specific extensions to the core *m3po* ontology. Since these extensions model information that is not present in the workflow model, they need to be added manually by a domain expert.

4.1 Functional Aspect

The functional aspect in a workflow model can be seen as a meta-concept defining the reusability characteristics of the modelling elements in the underlying aspects. In XPDL as well as in *m3po* the *Package* element type serves as the top-level concept in a process model, referencing on the one hand aspect specific properties and on the other hand comprises of concepts to describe generic workflow properties. Some properties introduced in version 2.0 of XPDL [20] to represent BPMN diagrams (including the graphical layout) have weak semantics. For example, the *Pool*, the *Participant* and the *Lane* elements are only of type *string*. In *m3po* the range of these properties are *Participants*, *ProcessTypes*, and *ParentLanes* respectively. The translation algorithm presented in the next section performs a type check when creating *Pool* instances by trying to match the string in the pool element with translated objects of type *Participant* or *ProcessType*. If no matches are found, the type of the pool element remains a string type, but a stronger typing can be added manually.

Choreography extensions We distinguish *Private-*, *Abstract-* and *PublicProcessTypes* in *m3po*. A *PrivateProcessType* describes activities that constitute the internal workflow. The translation of an XPDL model yields by definition to a *PrivateProcessType*. *AbstractProcessTypes* are used to model interface models, such as the choreography interface descriptions we extract by compacting an annotated *PrivateProcessType* as described in section 5. *PublicProcessTypes* are used to model collaborative processes, i.e. a composition of *AbstractProcessTypes*.

4.2 Control Aspect

The control aspect is concerned with the task ordering in a process model and the routing along those tasks. In XDPL as well as in the proposed ontologisation in *m3po*, the *Process* type constitutes the primary modelling element. It groups related activities, data, and resources together. The *Activity* class in *m3po* represent the reusable task behaviour in a process and may take one of the following types, a task, a route activity (a gateway in BPMN) that constrains the ordering of activities, a subflow (a reusable SubProcess in BPMN), a block activity (an embedded SubProcess in BPMN) or an event (cf. Fig. 5).

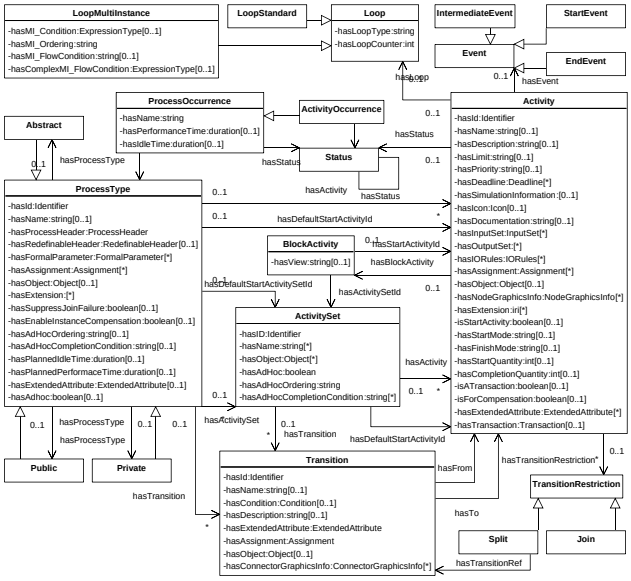


Figure 5: Concepts in the control aspect

The actual control flow in XPDL and *m3po* is modelled via explicit transitions between elements, including the routing activities modelling control flow constructs such as a *Loop* and a *Split*. This modelling paradigm is compatible with the behavioural definitions in BPMN, which as a graph based modelling notation relies on the modelling of connectors to define the control flow. Accordingly, the definition of control flow links in *m3po* is modelled with a *Transition* concept (cf. Fig. 5), representing arcs as a first class entity in the model, with associated *hasFrom* and *hasTo* properties.

Choreography extensions The annotations required in choreography models in regard to the control aspect are concerned with the parameterised views on activities. Process activities in *m3po* can be defined to be visible only to a specific partner by the provision of the *isVisibleTo* attribute. This relation is used in the compaction rules to create a choreography interface for a specific partner (cf. section 5). When the visibility relation is added to a model the respective activity is by definition member of an *AbstractProcessType*.

[5]The complete diagram can be found at: `http://www.m3pe.org/`.

4.3 Informational Aspect

The informational aspect deals with data production and data consumption, i.e. the data flow between tasks. We provide a direct mapping from all data types to the ontology. Simple datatypes are directly modelled as WSML datatypes which inherit the type hierarchy from XSD Schema. Complex datatypes are modelled similarly to XPDL.

XPDL and *m3po* offer means to model all common ways of data passing, either along the control flow, by defining an explicit data flow with data connectors or as standard by the global access to variables. The passing of data along the control flow can be achieved via *DataMapping* objects that are referenced from *Activities* and which define the direction of the data with a *hasFromParameter* and *hasToParameter* attribute. A *DataMapping* class can also include a logical expression to define a mapping condition. A more common way to pass data is the modelling of message flows. For the modelling of internal message passing a message flow can be defined on the activity level, connecting a *MessageType* with a *hasFrom* and *hasTo* attribute to an *Activity*. To model collaborative Web service based processes XPDL and *m3po* offer message passing on the *PackageType* level between *Pool* types. However, whereas messages in XPDL

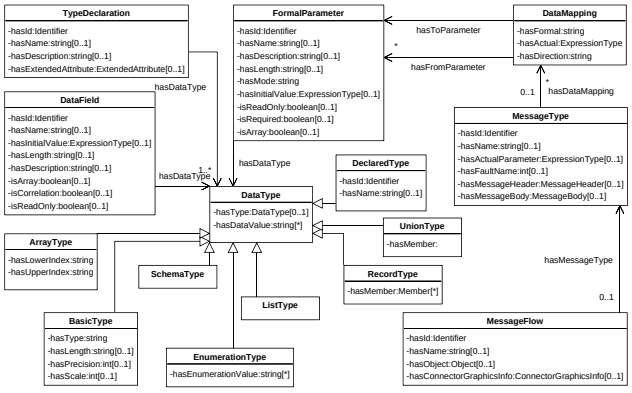


Figure 6: Concepts in the informational aspect

are only referenced by an external identifier, in *m3po* such identifiers can also refer to external message ontologies. Since the messages in our motivating example are defined by RosettaNet PIP schemas, we can reuse an existing RosettaNet ontology to indicate the exchanged messages in each activity.

Choreography extensions Data transfer in collaborative processes is achieved by defining the sequence and the conditions in which *messages* are exchanged. The modelling of messages as described above is already offered by XPDL for workflow modelling. The only difference for collaborative processes is that messages additionally need to be referenced from *collaborationRoles*.

4.4 Organisational Aspect

The organisational aspect defines *who* is responsible for performing a task in a workflow and gives the modeler means to assign constraints on the agent responsible for carrying out a task. The organisational aspect in XPDL is only weakly defined; a process participant is simply a token of the following types: resource set, resource, organisational unit, role, human, or system. In *m3po* these types are modelled as stand-alone concepts based on representations in the Suggested Upper Merged Ontology (SUMO)[6], a richly axiomatised formal ontology. However, since the internal role model is of little interest for the choreography generation presented in this paper we refer to [10] for more details about the organisational modelling in *m3po*.

Choreography extensions The modelling of choreographies requires a role model different to the internal roles defined in the workflow model. The *collaborationRole* defines the observable behaviour that a party exhibits when collaborating with other parties in the external process. The "Customer" role in our example is associated with requesting a quote from a "Supplier" role. To give one partner the possibility to define restrictions on the functionality that must be provided by other partners in a choreography, the ontology includes *partnerLinks*. Each *partnerLink* is characterised by an associated *collaborationRole* that has to be played by the collaboration partner.

4.5 Operational Aspect

The operational aspect deals with the modelling of external applications, managed or invoked by the WfMS. These applications can be different in nature, but their technical details are not required and always abstracted in the workflow model. Although XPDL offers modelling concepts for different application types such as EJBs, Java Objects, XSLT scripts, Forms and Business Rules, the most important application type in the context of this paper are Web services. The semantics of all former application types in *m3po* are kept similar to XPDL, but the modelling of Web Service is enriched as follows (cf. Fig. 7[7]). First, the associated input and output messages are referenced as *Message* types. Second, *PartnerLinks* and *PartnerLinkTypes* are explicitly linked, similarly to the modelling of Web Services in BPEL. In *m3po* the *PartnerLink* modelled in the *PackageHeader* references these *PartnerLinkTypes* and defines which role is taken by the process itself and which role is taken by a *Partner*. In this way, the *PartnerLinkType* in *m3po* describes a contract between two partners in terms

[6] See `http://www.ontologyportal.org/`

[7] Please note that the activity and process concept are shown in the figure for illustrative purpose, although by definition they do not belong to the operational aspect.

of their roles (which are also related to a *Pool*) and the corresponding WSDL *PortTypes* the partners have to provide.

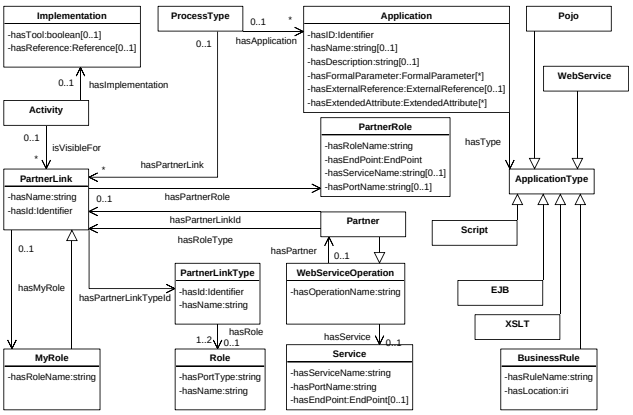


Figure 7: Concepts in the operational aspect

5 Compaction Rules

In this section we present compaction rules that extract a sub-process (choreography interface) from an annotated workflow including all activities that receive or send messages to a particular partner (and are annotated as such), but hiding all irrelevant information for this specific partner cooperation. The problem of extracting views on workflows is well-established and a number of solutions have been proposed in literature [4, 18, 5]. We adopt the solution proposed in [4] and adapt it to the ontological model of *m3po*. We define two compaction rules to identify and remove private control dependencies within a Sequence, OR, XOR, and AND block. The compaction rules introduced in this section are recursively applied to a workflow model until no nodes can be hidden anymore.

Definition 5.1 (Workflow Graph)**.** *A directed workflow graph is a tuple $\mathcal{G}(\mathcal{O}, \mathcal{F})$, where $\mathcal{O}$ is a finite set of object nodes (e.g. activities) according to the XPDL metamodel and $\mathcal{F} \subseteq \mathcal{O} \times \mathcal{O}$ is a control flow relation, i.e. a set of sequence flows connecting objects.*

The workflow graph represents the translated XPDL model in m3po. We now define the annotated workflow graph, corresponding to an *m3po* model where activities (objects) are annotated with visibility properties as defined in the choreography extensions in section 4.2.

Definition 5.2 (Annotated Workflow Graph)**.** *An annotated workflow graph $\mathcal{G}'(\mathcal{O}', \mathcal{F}, visibility)$ is a workflow graph $\mathcal{G}$, where for every $o' \in \mathcal{O}', o' \equiv o \in \mathcal{O}$ a function $visibility : \mathcal{O} \rightarrow \{private, public\}$ assigns the visibility of an object o in the graph to $private$ or $public$.*

Definition 5.3 (Sequence rule)**.** *Given the annotated workflow graph $\mathcal{G}'(\mathcal{O}', \mathcal{F})$ where $o_i, o_j, o_k \in \mathcal{O}'$ and o_j has $visibility$ $private$ and o_i, o_k has $visibility$ $public$ and $f_{ij}(o_i, o_j), f_{jk}(o_j, o_k) \in \mathcal{F}$ then a collaborative graph $\mathcal{G}_c(\mathcal{O}_c, \mathcal{F}_c)$ compacted by o_j is defined as a set of places $\mathcal{O}_c = \mathcal{O}' - \{o_j\}$ and $\mathcal{F}_c = \mathcal{F} - \{f_{ij}(o_i, o_j) f_{jk}(o_j, o_k)\} \cup \{f_{ik}(o_i, o_k)\}$.*

This rule shows that if we dispose of a private activity o_j followed by any other activity o_k, then we eliminate o_j and all the flows connecting to o_j and we create a new flow linking the $f_{ik}(o_i, o_k)$. This rule also applies to inclusive and exclusive branching (OR and XOR-splits) where one branch is replaced by an arc to the respective join activity.

Definition 5.4 (AND rule)**.** *Given the annotated workflow graph $\mathcal{G}'(\mathcal{O}', \mathcal{F})$ where $o_i, o_j, o_k, o_l \in \mathcal{O}'$ and o_j has $visibility$ $private$ and o_i, o_k has $visibility$ $public$ and $f_{ij}(o_i, o_j), f_{ik}(o_i, o_k), f_{kl}(o_k, o_l), f_{jl}(o_j, o_l) \in \mathcal{F}$ then a collaborative graph $\mathcal{G}_c(\mathcal{O}_c, \mathcal{F}_c)$ compacted by o_j is defined as a set of places $\mathcal{O}_c = \mathcal{O}' - \{o_j\}$ and the set of transitions $\mathcal{F}_c = \mathcal{F} - \{f_{ij}(o_i, o_j) f_{jl}(o_j, o_l)\}$.*

If a private activity o_j is in a parallel branch to other activities and immediately preceded by an AND-split and directly succeeded by an AND-join, indicating that all its paths should be executed in parallel, it is removed and all incoming and outgoing arcs are removed. If there is only one other branch it becomes a simple sequence, otherwise a parallel branch reduced by one branch.

If we apply these rules to the annotated workflow graph of our motivating example (cf. the output of our translation tool on the right side of Fig. 3) we obtain a model in *m3po* that describes a control flow as depicted in Fig. 8.

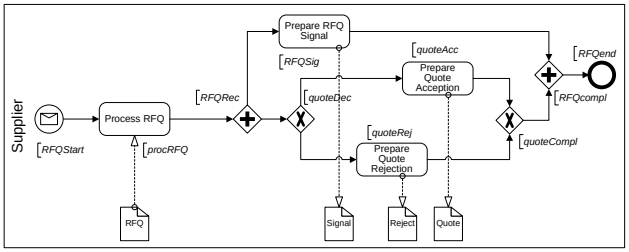


Figure 8: Compacted supplier's choreography interface

6 Mapping *m3po* to an executable choreography interface model

The compacted choreography interface model in *m3po* as presented in the previous section is not meant for execution per se, but it can be translated to an executable choreography model. The resulting definition can then be used to control the message exchange in a collaboration, in our

case between the "Supplier" and the "Customer". In this paper we propose as a target model the state-based WSMO choreography model which describes the behaviour of a service from one role instance defining send and receive events (i.e. choreography interface) in Web service communications. The benefit, beyond the expressivity of the WSMO choreography model (and thus the relative ease of the mapping), is the possibility of combining provided and provider interface [7] within one model. In our case we model the choreography interface of the "Supplier". As such, the generated model can be used to control the message exchange of the "Supplier" in a semantic service-oriented architecture. However, if needed, the "Customer" interface can be generated similarly to the provider behaviour.

6.1 WSMO choreography

WSMO choreography models [19] are based on Abstract State Machines (ASMs) [3]. A signature, defined by a schema of the information interchanged, defines predicates and functions to be used in the description. Ground facts which are instances of concepts and relations defined by imported ontologies specify the underlying database states. A set of transition rules in a guarded transition denote the state changes with regard to the evolution of the information space throughout the consumption of the functionality of the Web service. State changes are described by the creation of new instances or changes to attribute values of objects. Transition rules have one of the following forms:

- $if\ \varphi\ then\ R; end\ if$
- $forall\ x\ with\ \varphi\ then\ R; end\ forall$
- $choose\ x\ with\ \varphi\ then\ R; end\ choose$

The condition φ is a Boolean term and the update R is a set of transition rules. The update part which can be a primitive state change, like add, delete, or update, defines all changes on the information space from an initial state S to S' where the condition φ is satisfied. More complex transition rules can be defined with the help of if-then, forall and choose rules. In a given state, all updates of transition rules whose condition is true, fire simultaneously (order does not matter). If the updates are consistent it yields to the next state.

6.2 Mapping m3po models to WSMO choreographies

We assume a process model expressed in *m3po* to be a directed control-flow graph. This assumption is valid for all translations of BPMN-based XPDL models and for the majority of proprietary workflow models exported to XPDL. We further assume a graph modelled with one initial and one final activity (although theoretically one could model a valid graph in *m3po* with multiple end activities). Transitions between activities denote successor relations and different gateway types such as the AND, XOR or OR splits and joins denote whether interactions must be executed concurrently (AND-split), or whether exactly one branch is chosen (XOR-split) or whether at least one branch is chosen (OR-split).

We will illustrate the mappings based on the extracted choreography model from the internal process as described in section 5. Fig. 8 shows the choreography of our *Supplier* with a *Customer*. It is the abstracted view of the entire workflow as depicted in our motivating example with the "Check Product Availability" hidden because of its annotation as a private activity and the "Manage Account" hidden because of its visibility to the *Transaction Bank* role only. As shown in Fig. 8 all successors of the initial "Process RFQ" must be executed, since they follow an AND-split gateway. The branches that correspond to these successors eventually join in an AND-node before the process ends. However, one of the two parallel branches diverges in an XOR-split that means only one of the two activities "Prepare Quote Rejection" or "Prepare Quote Acception" will be enabled. In order to illustrate the mapping rules we have marked the gateway nodes (such as the *quoteDecision* XOR-gateway) in Fig. 8 with comprehensible names. In the originating *m3po* model they are identified by a URI. In order to translate the compacted choreography interface model in *m3po* to the executable choreography specification we define the state signature of the model as follows:

- The set of activities in the process model, including the gateway activities representing the XOR, OR and AND split and join, respectively.
- The set of *ActivityOccurrences* of a process instance with the status attribute denoting valid states. For the mapping we restrict the state space to three status, *active*, *inactive* and *completed*. *ActivityOccurrence* are created for all activities during the compaction process. The *inactive* state is considered to be the standard state and is considered to be true for all *ActivityOccurrences*. We introduce a boolean proposition $active$ returning true for all *activated ActivityOccurrences*. This proposition is used to determine which branch has been chosen (activated) in an OR-split. The decision can be based on the evaluation of a logical expression in the ontology itself or added by the environment.
- The set of transitions connecting the activities. For simplicity we introduce a binary relation $transition$ taking the values of the *hasTo* and *hasFrom* attribute in the Transition class.

Based on this state signature the actual state in the WSMO choreography specification is determined by the truth value of the following propositions:

- The set of valid *ActivityOccurrence* instances. All activities compacted from a workflow model are by definition valid instances in the choreography interface.
- The set of completed *ActivityOccurrence*. If the proposition $completed$ evaluates to true it means the particular *ActivityOccurrence* instance has been executed, otherwise it is in an active state. In case of XOR or OR splits we use this proposition to determine which branch has been chosen by the environment.
- The set of instances of the predicate $transition$. An instance $transition(actOcc1, actOcc2)$ evaluates to true if there is an arc from *actOcc1* to *actOcc2* and in case there is a condition connected to the arc it evaluates to true.

In the following we define a translation T of an *m3po* ontology graph G to a set of WSMO Choreography transition rules. The modifier $add(b)$ is used to change the truth value of the proposition b to true as described in the previous section. Such a transition rule where b becomes true ensures that the status of an *ActivityOccurrence* is set to completed.

$$\mathrm{T}(G) = if\ a \wedge \neg b \wedge transition(a, b) \wedge active(b)\ then\ add(b)$$

This transition rule ensures the completion of an *ActivityOccurrence* b if there is an arc from a to b and a is an OR-split gateway activity.

$$\mathrm{T}(G) = if\ a_1 \wedge\ ... \wedge a_n \wedge \neg b \wedge transition(a_1, b) \wedge ... \\ \wedge transition(a_n, b)\ then\ add(b)$$

This transition adds a completion status to an *ActivityOccurrence* b if there is an arc from each a_1 ... a_n to b and b is an AND-join gateway activity.

$$\mathrm{T}(G) = if\ a \wedge \neg b \wedge transition(a, b)\ then\ add(b)$$

Further, an *ActivityOccurrence* status is updated to completed if there is an arc from a to b, but a is not an OR-split gateway and b is not an AND-join gateway activity.
If we apply these three rules to our compacted model we obtain the following transition rules in a WSMO choreography model. The arcs originating in an OR-gateway result in the following rules:

$$if\ quoteDec \wedge \neg quoteAcc \\ \wedge transition(quoteDec, quoteAcc) \\ \wedge active(quoteAcc)\ then\ add(quoteAcc) \\ if quoteDec \wedge \neg quoteRej \\ \wedge transition(quoteDec, quoteRej) \\ \wedge active(quoteRej)\ then\ add(quoteRej)$$

For the arcs ending in the AND-join gateway in our example the following rule is obtained:

$$if\ RFQSig \wedge quoteCompl \wedge \neg RFQCompl \\ \wedge transition(quoteCompl, RFQCompl) \\ \wedge transition(RFQSig, RFQCompl) \\ then\ add(RFQCompl)$$

Applying mapping rule 6.2 to the remaining arcs yield to the following transition rules in the choreography model:

$$if\ RFQStart \wedge \neg procRFQ \wedge transition(RFQStart, procRFQ) \\ then\ add(procRFQ)$$
$$if\ procRFQ \wedge \neg RFQRec \wedge transition(procRFQ, RFQRec) \\ then\ add(RFQRec)$$
$$if\ RFQRec \wedge \neg RFQSig \wedge transition(RFQRec, RFQSig) \\ then\ add(RFQSig)$$
$$if\ RFQRec \wedge \neg QuoteDec \wedge transition(RFQRec, quoteDec) \\ then\ add(quoteDec)$$
$$if\ quoteAcc \wedge \neg quoteCompl \wedge transition(quoteAcc, quoteCompl) \\ then\ add(quoteCompl)$$
$$if\ quoteRej \wedge \neg quoteCompl \wedge transition(quoteRej, quoteCompl) \\ then\ add(quoteCompl)$$
$$if\ RFQCompl \wedge \neg RFQEnd \wedge transition(RFQCompl, RFQEnd) \\ then\ add(RFQEnd)$$

6.3 Executing mapped choreographies

As explained before, the generated WSMO choreography document can be used to control the execution of the Web service message exchange patterns within a semantic service-oriented architecture such as WSMX [9]. Such architectures exist and the generated choreography can be deployed in them. A detailed description of the deployment of the model and the execution semantics is beyond the scope of this paper. The interested reader is referred to [11].

Note that our approach and model is not restricted to WSMO choreographies. We can target arbitrary choreography languages by adding appropriate declarative mappings. We have selected WSMO choreographies as demonstration target because of the expressive formalism and the existence of execution platforms.

7 Conclusion

State-of-the-art workflow management systems do not support the (semi-)automatic construction of choreography interfaces from existing workflows. These choreography interfaces are needed in cross-organisational collaborations which are becoming ever more automated through service-oriented computing. Being not able to create them automatically from the internal workflows causes not only additional labour, but also synchronisation issues that compromise the consistency between the workflow and the choreography model in its execution. We presented an approach

to automatically generate partner-specific choreography interfaces from internal workflows. We use an integrated ontology *m3po* for representing business workflows and choreographies, based on the XPDL metamodel. We presented declarative mapping rules from XPDL to our ontology which we implemented in a conversion tool. Based on light-weight manual choreography annotations on the automatically populated ontology that capture parameterised view properties, we presented compaction rules that generate per-partner views on the workflow model. We showed how to generate executable choreography specifications from these generated partner views, again using declarative mappings. In particular, we presented a mapping to WSMO choreographies as demonstration target because of the expressivity of the formalism and the existence of execution platforms. The resulting models can be used directly for the execution of a collaborative process or exchanged by business partners to negotiate a global choreography and to adopt their Web service interfaces accordingly.

Acknowledgements This material is based upon works jointly supported by the Science Foundation Ireland under Grant No. SFI/08/CE/I1380 and the European Union under the SUPER project (FP6-026850).

References

[1] A. Alves *et al.* Web Services Business Process Execution Language Version 2.0. OASIS Standard, OASIS, Apr. 2007.

[2] R. Bobrik, M. Reichert, and T. Bauer. View-Based Process Visualization. In *Proceedings of the 5th International Conference on Business Process Management*, pages 88–95, Brisbane, Australia, 2007.

[3] E. Börger and R. Stärk. *Abstract State Machines: A Method for High-level System Design and Analysis*. Springer-Verlag, 2003.

[4] I. Chebbi, S. Dustdar, and S. Tata. The view-based approach to dynamic inter-organizational workflow cooperation. *Data & Knowledge Engineering*, 56(2):139–173, 2006.

[5] D. K. W. Chiu, S. C. Cheung, S. Till, K. Karlapalem, Q. Li, and E. Kafeza. Workflow View Driven Cross-Organizational Interoperability in a Web Service Environment. *Information Technology and Management*, 5(3-4):221–250, 2004.

[6] J. de Bruijn, H. Lausen, A. Polleres, and D. Fensel. The web service modeling language: An overview. In *Proceedings of the 3rd European Semantic Web Conference*, Budva, Montenegro, June 2006.

[7] R. M. Dijkman and M. Dumas. Service-oriented Design: A Multi-viewpoint Approach. *International Journal of Cooperative Information Systems*, 13(4):337–368, Dec. 2004.

[8] R. Eshuis and P. Grefen. Constructing customized process views. *Data & Knowledge Engineering*, 64(2):419–438, 2008.

[9] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX – A Semantic Service-Oriented Architecture. In *Proceedings of the 3rd IEEE International Conference on Web Services*, pages 321 – 328. Orlando, FL, USA, 2005.

[10] A. Haller, E. Oren, and P. Kotinurmi. m3po: An Ontology to Relate Choreographies to Workflow Models. In *Proceedings of the 3rd International Conference on Services Computing*, pages 19–27, Chicago, IL, USA, 2006.

[11] A. Haller, J. Scicluna, and T. Haselwanter. WSMX Choreography. WSMX Working Draft, DERI, 6 2005.

[12] S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation*. International Thomson Computer Press, 1996.

[13] N. Kavantzas *et al.* Web Services Choreography Description Language. Technical report, Nov. 2005.

[14] A. Martens. Consistency between Executable and Abstract Processes. In *Proceedings of the International Conference on e-Technology, e-Commerce and e-Service*, Hong Kong, China, Mar. 2005.

[15] M. P. Papazoglou and D. Georgakopoulos. Service-Oriented Computing. *Communications of the ACM*, 46(10):25–28, 2003.

[16] K. H. Park and J. Favrel. Virtual enterprise – Information system and networking solution. *Computers & Industrial Engineering*, 37(1-2):441–444, 1999.

[17] D. Roman, M. Kifer, and D. Fensel. WSMO Choreography: From Abstract State Machines to Concurrent Transaction Logic. In *Proceedings of the 5th European Semantic Web Conference*, pages 659–673, Tenerife, Spain, 2008.

[18] K. A. Schulz and M. E. Orlowska. Facilitating cross-organisational workflows with a workflow view approach. *Data & Knowledge Engineering*, 51(1):109–147, 2004.

[19] J. Scicluna, A. Polleres, and D. Roman. Ontology-based Choreography and Orchestration of WSMO Services. WSMO Working Draft v0.2, DERI, 2005.

[20] The Workflow Management Coalition. Workflow Standard Process Definition Interface – XML Process Definition Language. Technical Report WFMC-TC-1025, WfMC, Oct. 2005.

[21] H. Tran, U. Zdun, and S. Dustdar. View-Based Reverse Engineering Approach for Enhancing Model Interoperability and Reusability in Process-Driven SOAs. In *Proceedings of the 10th International Conference on Software Reuse*, pages 233–244, Beijing, China, 2008.

[22] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.