DERI – DIGITAL ENTERPRISE RESEARCH INSTITUTE

A process ontology to represent semantics of different process and choreography meta-models

Armin Haller and Eyal Oren

DERI Technical Report 2006-02-03
February 2006

DERI Galway
IDA Business Park
Galway, Ireland
www.deri.ie

DERI Innsbruck
Technikerstrasse 21a
Innsbruck, Austria
www.deri.at

DERI Korea
Yeonggun-Dong, Chongno-Gu
Seoul, Korea
korea.deri.org

DERI Stanford
Serra Mall
Stanford, USA
www.deri.us

DERI – DIGITAL ENTERPRISE RESEARCH INSTITUTE

DERI Technical Report
DERI Technical Report 2006-02-03, February 2006

A process ontology to represent semantics of different process and choreography meta-models

Armin Haller[1] Eyal Oren[1]

Abstract. Currently external business process descriptions (choreographies) are disconnected from the internal processes (workflows), leading to several problems. Directly mapping internal to external processes requires a quadratic amount of mappings; an intermediate ontology reduces the amount of necessary mappings but is not trivial to construct, due to the variety in workflow metamodels. In this paper we introduce our multi metamodel process ontology (*m3po*), which is based on various existing reference models and languages from the workflow and choreography domain. This ontology allows the extraction of arbitrary choreography interface descriptions from arbitrary internal workflow models. We also report on an initial validation: we translate an IBM Websphere MQ Workflow model into the *m3po* ontology and then extract an Abstract BPEL model from the ontology.

[1]Digital Enterprise Research Institute, National University of Ireland, Galway.

Acknowledgements: This material is based upon works supported by the Science Foundation Ireland under Grant No. 02/CE1/I131.

Contents

1 Introduction

Organisations have long used process modelling to describe the dynamic behaviour of their business. Workflow Management Systems (WfMSs) are commonly applied for process modelling and allow to describe and execute business processes [10]. With the advent of Service Oriented Computing [18] organisations started to expose their business functionality explicitly as reusable and composable services. For using these services organisations offer choreography interfaces (also called public processes, abstract processes, or provider behaviour [9, 24]), stating conversational patterns in which the services can be consumed.

A fundamental lack in current choreography frameworks such as the Abstract Business Process Execution Language (BPEL) [24] and Web Service Choreography Description Language (WS-CDL) [14], is the disconnection between external choreography interfaces and internal workflow descriptions. Conceptually, a choreography interface can be regarded as an abstracted view on a business process (c.f. [6, 7, 9, 21]), but current choreography frameworks ignore this dependency relation.

The disconnection between choreography interfaces and workflow definitions leads to two problems: choreography interfaces have to be manually synchronised with workflow descriptions, and their consistency with regard to the internal process can not be verified automatically. A major obstacle in connecting internal processes and external choreographies is the variety in existing workflow languages, workflow metamodels, and choreography languages [2, 19, 23]. Directly translating from workflow languages to choreography languages would require n^2 mappings, for each combination of workflow and choreography language.

2 Motivating Example

We will now illustrate the problems that companies face when designing collaborative business processes with an example request-for-quote (RFQ) process.

2.1 Current situation

An automotive parts vendor implements and executes his internal processes with IBM Websphere MQ Workflow[1]. One of the vendor's processes concerns the processing of requests for quotes. Figure 1 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.

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.

The graphic in figure 1 only shows the behavioural aspect and parts of the informational aspect. Organisational and operational properties of the model, such as the role assignment of the manual activity *check-product-availability*, are not visible in the figure, since they are defined within activities.

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

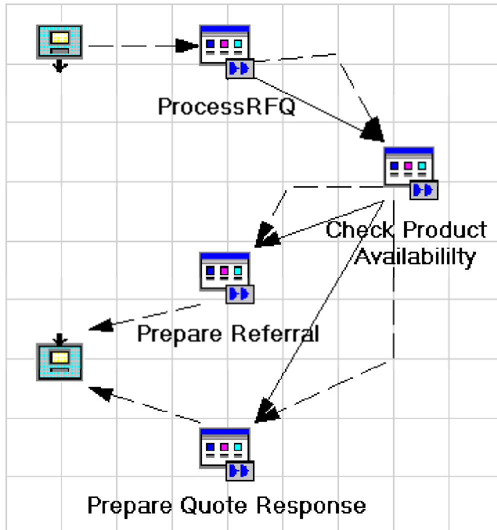

Figure 1: IBM MQ Workflow RFQ

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 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 confirm to industry standards this public process should conform to the standardised RosettaNet choreography interface PIP 3A1[2]; which describes a request for quotation.

Figure 2 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 vendor's choreography is formed by the black activities in his swimlane and the customer's choreography by the black activities in his swimlane respectively.

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; studies show that around 50% of the RosettaNet implementation effort focuses on combining the private process and the PIP public process, indicating a low level of automation [8].

In order to automatically extract the choreography interface, the internal business process has to be extended by information specific to external processes, such as message transfer, a collaboration role model, and the direction of the communication. Subsequently the model should be extracted to a choreography descriptions language. These features are currently not offered by MQ Workflow or any other workflow management system.

[2] http://www.rosettanet.org/PIP3A1.

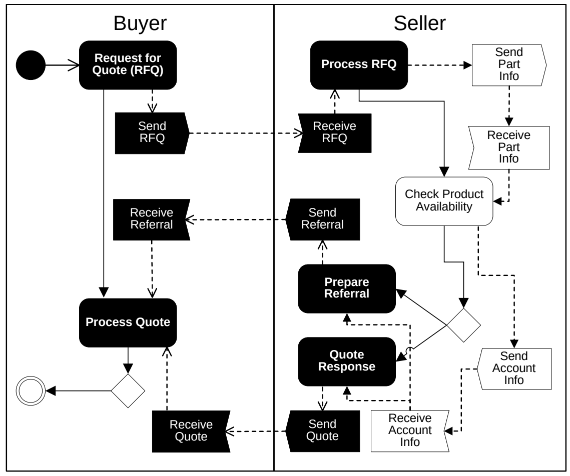

Figure 2: External Process (RosettaNet PIP)

3 Approach

Figure 3 describes our approach to connect workflows and choreography descriptions. We develop an intermediate unifying workflow ontology that can represent arbitrary workflows, thus reducing the amount of required mappings to $2n$. This ontology can be used to represent internal process models, and from the ontology choreography interfaces (corresponding to the internal process model) can be extracted.

In this paper we focus on developing the ontology. We construct the ontology based on an analysis of existing models, and we validate the representational power of this ontology by first capturing an exemplary existing model, and extracting an exemplary choreography interface from the ontology.

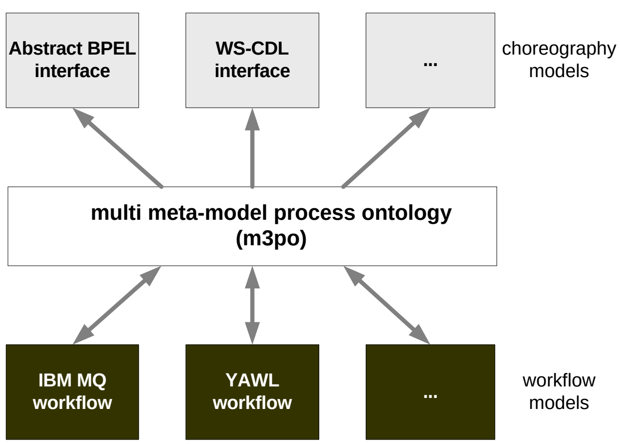

Figure 3: Connecting workflow models to extract choreography interfaces

3.1 Hypothesis

Our hypothesis is that constructing the ontology through a careful analysis of existing reference models and workflow languages, guarantees the representational width of the ontology, i.e. that all existing workflow models can be represented and all existing choreography interfaces can be extracted from it.

To verify this hypothesis, we first analyse the most prominent existing reference models, and analyse their features and representational power. The analysis is performed on seven evaluation criteria: first we consider the support for vertical integration (the ability to relate internal and external process models) and horizontal integration (the ability to map and relate different WfMSs, to reduce the number of necessary mappings). Then we consider the support for the five key workflow aspects [13], widely recognised as essential workflow characteristic: the functional[3], behavioural, informational, organisational, and operational aspects. And finally we compare the support for choreography specific extensions, specifically the support for message definitions and message passing, and collaboration role-models.

3.2 Analysis

We summarise the support of workflow and choreography features in existing reference models and languages, as shown in table 1.

	vert.	horiz.	behav.	infor.	org.	oper.	chor.
XPDL	−	±	±	+	±	+	+
PSL	−	+	±	±	−	−	−
YAWL	−	−	+	+	−	±	−
BPEL	±	−	±	+	−	±	+
WS-CDL	−	−	±	+	−	−	+

Table 1: workflow and choreography features

We include the following models in our analysis: XPDL [25]: the standard for syntactical exchange of workflow models; PSL [12]: a formal ontology that can represent the semantics of workflow models and enables semantics-preserving exchange of models; YAWL [1]: the behaviourally most complete workflow language with direct support for all workflow patterns; BPEL [24]: the most prominent (executable) web orchestration language; and WS-CDL [14]: the most prominent multi-party choreography language.

Vertical integration is concerned with the integration of internal and external process models, it includes workflow views [7], abstraction levels, and visibility of processes and activities. Abstract BPEL describes choreographies, but cannot indicate visibility and is disconnected from executable BPEL. XPDL has private and public processes, but they cannot be defined on the activity level, nor can the visibility be parameterised for a participant. WS-CDL only describes choreographies and has no notion of internal processes, whereas PSL and YAWL do not support external process models.

[3] the functional aspect is not separately analysed

Horizontal integration denotes the ability to deal with multiple WfMSs. PSL offers a formal ontology that can express semantical differences between systems. XPDL offers workflow interoperability on a syntactical level, but cannot express semantical differences. YAWL, BPEL, and WS-CDL do not address integration, but are stand-alone models.

In the *behavioural* aspect, YAWL naturally supports all control-flow patterns. BPEL supports more advanced patterns, whereas XPDL, PSL, and WS-CDL support only the basic control-flow constructs (although PSL allows arbitrary extensions). Constraint-based approaches [3] are only supported by PSL.

In the *informational* aspect, XPDL, YAWL, BPEL, and WS-CDL support data type definitions and data passing. PSL does not cover data passing or typing.

In the *organisational* aspect, XPDL supports the usage of external resource definitions (it supports the terms, but relations cannot be included). PSL, YAWL, BPEL, and WS-CDL do not include an organisational model.

In the *operational* aspect, XPDL offers various invocation methods and styles. YAWL and BPEL use Web services for invoking operations. PSL and WS-CDL do not cover this aspect.

XPDL, BPEL, and WS-CDL support *choreography specific* aspects, whereas PSL and YAWL not.

4 Related Work

Our work is most closely related to several approaches to views on process models, i.e. [6, 7, 20, 21].

Chebbi et al. [6] propose a view model with cooperative activities that can be partially visible for different partners, but the approach requires n^2 mappings and does not consider data (message transfer).

Chiu et al. [7] present a meta-model that includes cross-organisational communications, defining message transfer and direction. The abstracted view is however limited to sequential activities, the approach is specific to one workflow modelling tool, and does not offer integrated choreography extraction.

Schulz/Orlowska [21] introduce a state-transition approach that binds states of private workflow tasks to a corresponding view-task; they identify mappings in the conceptual architecture, but do not describe how to integrate different workflow models, and abstract the data aspect completely.

Sayal et al. [20] 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.

Several approaches address interoperability issues between workflow management systems, such as Mobile [13], Meteor [22], and CrossFlow [11]. Unfortunately these approaches require a pre-established partner agreement on the semantics of the process models, and they do not target current choreography standards.

(author?) [15] describes consistency verification between executable BPEL and Abstract BPEL. However, the approach does not extract interfaces but only verifies them, and does not tackle the generic language mapping problem.

5 Ontology

The multi metamodel process ontology (*m3po*) is based on two principles: to incorporate and unify the different existing workflow metamodels and workflow reference models, and to provide the necessary properties for extracting choreographies from internal business processes. We have combined the most elaborate existing reference metamodels and extended the *m3po* with choreography-specific information. The ontology focuses on integrating collaboration agreements with internal processes; outside the scope are protocol-level agreements such as RosettaNet RNIF[4], which include validation rules, acknowledgement behaviour, security, and message transport [5, p. 368]; we do (partially) consider transaction management and message vocabulary.

In the following sections we describe the *m3po*, introduced per workflow aspect. The ontology is written in the web ontology language WSML [4], making the semantics of the process concepts formal and explicit — necessary given the different semantics of workflow meta-models. For readability reasons, we display the important concepts in UML class diagrams[5]; the full ontology including the axiomatisation of its semantics can be found at `http://m3pe.org/ontologies/m3po.wsml`. We illustrate each section with example snippets[6] that represent our earlier introduced example.

5.1 Functional and Behavioural Aspect

The functional and behavioural aspects of the *m3po* are shown in figure 4.

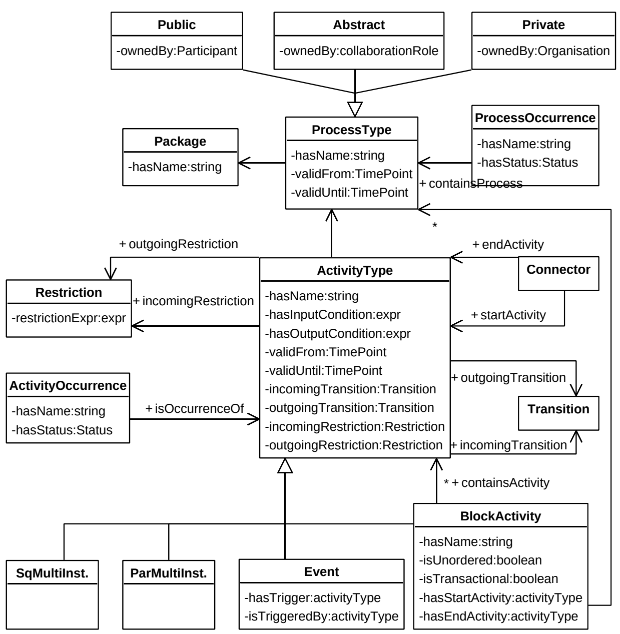

Figure 4: Functional and behavioural aspects

[4]`http://www.rosettanet.org/RNIF`.

[5]some concepts from the ontology are omitted in the UML class diagrams

[6]the complete ontology instance for that example is available at `http://m3pe.org/ontologies/rfq.wsml`.

5.1.1 Workflow related (behavioural)

An *activityType* is the primary concept in the ontology; it can represent a reusable behaviour in a process, a triggered event, a routing construct that constrains the ordering of other *activityTypes* or a task with pre- and postconditions to model constraint-based workflow specification languages [10].

A *processType* groups related activities, data, and resources together. The distinction between a *processType* and a *processOccurrence* is similar to workflow types and workflow instances in [13]. At execution (runtime), a *processOccurrence* represents an actual task to be carried out. Each *processOccurrence* is an instance of a *processType*. Multiple instances of a *processType* might exist at the same time. Listing 1 shows a snippet including instances of the *processType* and *activityType* from our motivating example: the overall *request-for-quote* process and the *process request-for-quote* activity.

```
instance rfqpw memberOf publicProcessType
  hasName hasValue "RFQ Processing Workflow"
  hasPartnerLink hasValue buyerSellerRelation

instance rfq memberOf activityType
  hasName hasValue "Process Request for Quote"
  hasTask hasValue saveDataInDatabase
```

Listing 1: Example process- and activity-types

The explicit modelling of runtime execution using *processOccurrences* and *activityOccurrences* is similar to the modelling style of PSL. The semantics of ordering constraints is given on the occurrence trees of activities and processes: a particular constraint restricts the allowed (runtime) occurrence trees. The explicit modelling of occurrences is also used for the state-transition characteristics of processes and activities. The ontology includes common process states (from different WfMS), such as *active, suspended, resumed, cancelled, aborted, completed* etc. (not shown in the diagram).

To allow hierarchical composition of *activityTypes* and *processTypes* the ontology includes a *blockActivity* concept. Hierarchical activities or composite processes can be represented using the *containsActivity* and *containsProcess* attribute.

To incorporate the prevalent activity-based models [10] the ontology includes *connectors*, which model explicit control-flow ordering. A common set of higher level control-flow constructs is provided for convenience, using the workflow patterns [2] as a reference model. Conditional expressions and various split and join restrictions are provided for basic and advanced branching and synchronisation patterns. *ParallelMultiInstantiation* and *sequentialMultiInstantiation* model structural patterns and patterns involving multiple instances. The *isUnOrdered* property on *blockActivities* models a non-deterministic ordering of tasks.

For constrained-based modelling one can omit the *connectors* and instead define *input-* and *outputConditions* on the *activityTypes*. Listing 2 shows how to model explicit control-flow ordering on a snippet from our motivating example, stating that the *check-product-availability* has a control-flow and data-flow dependency on the *request-for-quote* activity.

```
instance cpa memberOf activityType
  hasName hasValue "Check Product Availability"
  hasSplitRestriction hasValue productAvailability

instance rfqToCpa memberOf {controlConnector, dataConnector}
  hasStartActivity hasValue rfq
  hasEndActivity hasValue cpa
```

Listing 2: Example control- and data-flow connectors

5.1.2 Choreography related (behavioural)

To extract choreographies from internal business processes, the *m3po* has to distinguish *private*, *abstract* and *publicProcessTypes*. *PrivateProcessTypes* are fully executable internal process models, *abstractProcessTypes* are used to model interface models, and *publicProcessTypes* are used to model collaborative processes. In the abstract and public processes, activities can be defined to be visible only to specific partners by the *isVisibleTo* attribute. Listing 3 shows two different visibility assignments from our example above; the *messageEvent* is visible for the buyer role (but also –implicitly– visible to the seller, as process-owner), whereas the *manualTask* is only visible to the seller. The visibility properties have to be added by a business analyst in the ontological model.

```
instance sourceNode memberOf {startEvent, messageEvent}
  hasName hasValue "Source Node"
  isVisibleFor hasValue buyer

instance checkStockApplication memberOf manualTask
  hasName hasValue "Check Availability in Material Management"
  hasPerformer hasValue warehouseman
  isVisibleFor hasValue seller
```

Listing 3: Manual task with visibility assignments

5.2 Informational Aspect

The informational aspect (see figure 5) is defined by the data and data-flow perspectives [13]. The data being provided in process models is categorised into control and production data, whereas control data is only relevant to the model itself (e.g. functional properties of the process model) and production data exists independently from the process model.

5.2.1 Workflow related (informational)

The *m3po* supports all common data patterns [19]: direct data passing (data-flow), indirect data-passing (shared data-store), and data-flow independent from control-flow, all including data transformations.

DataBagSchemas define data schemas for production as well as control data. They either point to an external reference (e.g. an XML Schema file) or to *dataElementDefinitions* with an *elementName* and an *elementType*. The allowed *types* correspond to the XML Schema datatypes[7]. The actual data values (i.e. *elementType*) passed in a process are defined on *activityOccurrences*.

[7] `http://www.w3.org/TR/xmlschema-2/`.

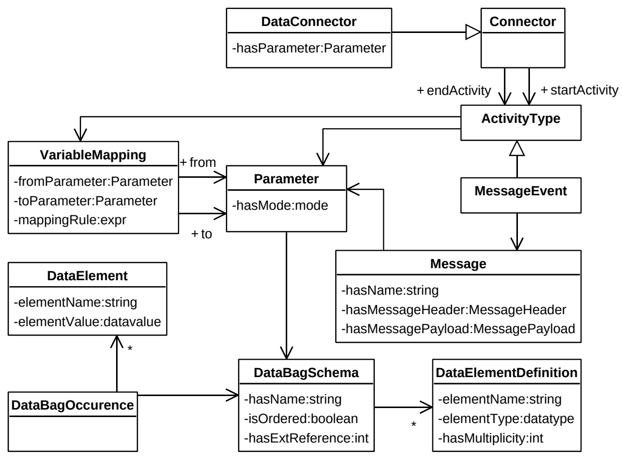

Figure 5: Informational aspect

```
instance processInput memberOf parameter
  hasDataBagScheme hasValue partDescription
  hasMode hasValue inParameter

instance partDescription memberOf dataBagScheme
  hasName hasValue "Part Description"
  hasDataElement hasValue rfqPartID
```

Listing 4: Example of explicit data passing

To model dataflow between activities, processes, and programs, *processTypes*, *activityTypes*, *programs* and *dataBags* can specify *parameters*. These parameters can have the modes *in-only*, *out-only*, or *in/out*. Listing 4 shows a part of the data definition in our motivating example, namely *dataBagSchemas* definitions (containing automotive parts) that belong to data passing *parameters*.

To allow data transformations, a mapping relation (*variableMapping*) can be defined if the *elementType* of an incoming parameter differs to the *elementType* of the outgoing parameter.

DataConnectors can pass data between *activityTypes* using *parameters*. This allows the modelling of a data passing mechanism that is independent from the control coordination.

To accommodate models which do not pass data explicitly, but share all data via a global data store [19] *processTypes* include a *parameter* attribute. If data is passed to the process with a *inParameter*, it is accessible to all activityTypes defined in this process. This method of data sharing is based on a shared a priori knowledge of the *elementName* and *elementType* of *dataElementDefinitions*. The *parameter* attribute also facilitates passing external data to and from the *processTypes* at instantiation and completion.

5.2.2 Choreography related (informational)

The fundamental modelling primitive in choreographies is the sequence and conditions in which *messages* are exchanged. The explicit representation of messages is usually not part of workflow models. Even if this fundamental approach to model data flow is possible in the internal model, it is only used to transfer data between *tasks*. In the case of a collaboration these messages are sent between *collaborationRoles* and contain a *messageHeader* (which stores control information about the *message*) and a *messagePayload* (the actual content of a *message*). Listing 4 shows the modelling of messages, containing *requests for quotes*; the message structure would be defined through *dataBagSchemas* or external *dataRepresentations*. Since message transfer is specific to external processes it would have been manually added to the model by a workflow engineer.

```
instance sourceNode memberOf {startEvent, messageEvent}
  hasName hasValue "Source Node"
  isVisibleFor hasValue buyer
  hasMessage hasValue rfqMessage

instance rfqMessage memberOf message
  hasName hasValue "Request for Quote Message"
  hasParameter hasValue processInput
  hasMessageHeader hasValue rfqMessageHeader
  hasMessagePayload hasValue rfqMessagePayload
```

Listing 5: Example use of *messages*

In the message-oriented approach the caller does not necessarily have to know the exact procedure that will be invoked, but instead creates a message of a specific format known to both roles, the *fromRole* and *toRole*. A grounding to a specific partner interface is not necessary.

The ontology further allows to define the visibility of *dataBags* to a specific *collaborationRole*. This gives the modeler the opportunity to restrict the access to data elements.

5.3 Organisational Aspect

The organisational aspect (see figure 6) defines who is responsible for carrying out a specific *task*. We include an adapted form of the organisational reference model introduced in [17].

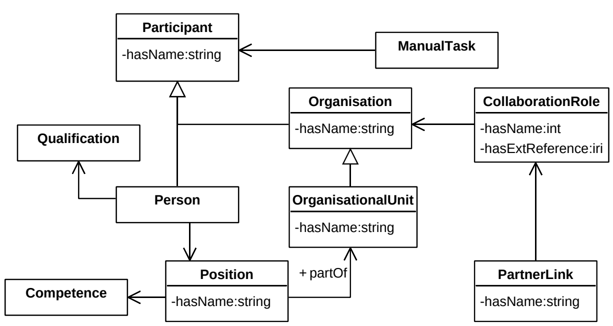

Figure 6: Organisational aspect

5.3.1 Workflow related (organisational)

A *participant* represents an organisational or human process resource. It is a named entity associated with a *manualTask*. The *belongsTo* attribute allows to hierarchically structure *organisationalUnits* and *persons*. *Competence* describes the possible actions a *participant* is permitted to perform. A *qualification* is a direct property of a *person*, and remains associated with the *person*, even if its *position* in the *organisation* changes. A *position* requires a *competence*, whereas many *persons* can meet these requirements with their *qualifications*. Holders of *positions* are granted the necessary authorities to perform the *tasks* associated with these *positions*. Groups of *positions* can be used to model for example temporary units (project teams) within an *organisation*.

5.3.2 Choreography related (organisational)

The modelling of choreographies requires an additional role model different to the internal role model; it has to be included manually. It should allow to specify the role of the *organisation* as a whole in an external business process. A *collaborationRole* defines the observable behaviour that a party exhibits when collaborating with other parties in the external process. 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.

To give one partner the possibility to impose restrictions on the functionality that must be provided by other partners in the external process, the ontology includes *partnerLinks*. Each *partnerLink* is characterised by an associated *collaborationRole* that has to be played by the collaboration partner. Listing 6 shows the role model from our example: the *seller* role is played by the automotive parts vendor, the *buyer* role is played by the car manufacturer.

```
instance buyerSellerRelation memberOf partnerLink
  hasName hasValue "Buyer/Seller Relation"
  hasRole hasValue {seller, buyer}

instance seller memberOf collaborationRole
  hasName hasValue "Automotive Parts Vendor"

instance buyer memberOf collaborationRole
  hasName hasValue "Car Manufacturer"
```

Listing 6: *PartnerLinks* and *collaborationRoles*

5.4 Operational Aspect

The operational aspect of the *m3po* is shown in figure 7. Current WfMSs have multiple ways to interact with their environment. Most systems distinguish between manual tasks performed by users, and automatic tasks, performed by automated computer programs.

The automatic tasks can be invoked in multiple ways, such as Web services or system applications. To represent this, operations are implemented by a *manualTask* or a *programTask*. The operational aspect can be seen as an interface to the external application, where the ontology models the execution properties.

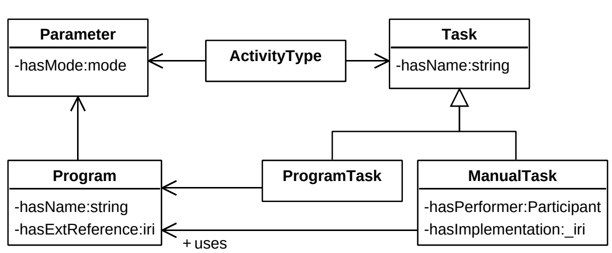

Figure 7: Operational aspect

The *parameter* attribute of *programs* controls the passing of the data required by the application. The presence of the according values is handled by the information perspective as described in section 5.2. *Programs* can be *asynchronous* or *synchronous*. The ontology allows to model different types of programs (e.g. *executableApplications* or *webServiceApplications*) as wrappers for any type of automatic task.

To indicate whether user interaction is required in the execution of a *program*, it can be associated with a *manualTask* to assign a *person* to the execution of the program. The *manualTask* is further used for manual operations that are performed by a human participant. It has an associated performer (i.e. *participant*) and a possible external reference to define data input forms.

5.5 Orthogonal Aspects

Several WfMSs allow rudimentary scheduling based on time. The *m3po* therefore includes *timeTriggerEvents* which are triggered if some time constraints are met. A specific *timepoint* or a *recurringCycle* (e.g. every Tuesday at 9am) can be set that will trigger the event. A *timeTriggerEvent* is itself an *activityType* and if it is used within the main flow it acts as a delay mechanism. If the event is used for exception handling it will change the normal flow into an exception flow. In order to allow the modeling of due dates and maximal duration time, *activityTypes* include according properties.

Integrity and failure recovery [13] is another orthogonal aspect taken into account in the ontology. *CompensationEvents* and *errorHandlingEvent* concepts are wrappers to compensate failed activities. Both events receive *dataBags* about the current state of the world and return data regarding the results of the compensation. *CompensationEvents* are also used to model transactional behaviour of processes. Transactions can be either compensatable, retrieable, or pivot [16]. The ontology allows to define *transactionalBoundaries* that associate *activityTypes* that should behave transactionally. If the transaction is *compensatable* all of its associated activities have to define compensatable event triggers. Every *activityType* within a pivot transaction has to define a *errorHandlingEvent* which triggers the termination of the process.

5.6 Summary

Applying the analysis from section 3.2, we see in table 2 that *m3po* is unique in the combination of workflow primitives and support for choreography-specific concepts. The ontology can act as a connecting ontology to integrate different workflow models (horizontal integration) and allows extraction of external process models (vertical integration).

	vert.	horiz.	behav.	infor.	org.	oper.	chor.
m3po	+	±	+	+	+	+	+

Table 2: workflow and choreography features of *m3po*

6 Validation

To validate our hypothesis we first need to show that indeed all metamodels can be represented, and then that arbitrary choreography interfaces can be extracted. We report on an initial result, namely the mapping from one WfMS (IBM MQ Workflow) into our ontology, and the mapping to one choreography language (Abstract BPEL), as shown in figure 8.

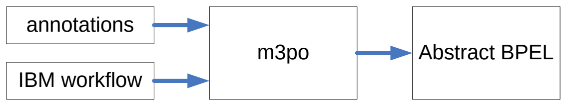

Figure 8: extracting a choreography interface

The mapping from the workflow model has been demonstrated already in the example snippets throughout section 5. The extracted choreography interface is shown in listing 7. 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, the organisational role model, 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">
4      <plnk:portType name="rfqpw"/>
5     </plnk:role>
6     <plnk:role name="buyer">
7       <plnk:portType name="rfqpwCallback"/>
8     </plnk:role>
9   </plnk:partnerLinkType>
10 </wsdl>
11
12 <process name="rfqpw"
13   targetNamespace="http://xmlns.oracle.com/BPELProcess1"
14   xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
15   xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
16   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
17   xmlns:lns="http://manufacturing.org/wsdl/purchase">
18  <partnerLink name="buyerSellerRelation"
19      partnerLinkType="lns:buyerSellerRelation"
20      myRole="seller"
21      partnerRole="buyer"/>
22  </partnerLinks>
23  <variables>
24     <variable name="rfqMessage" messageType="lns:rfqMessage"/>
25     <variable name="quoteMessage" messageType="lns:quoteMessage"/>
26     <variable name="referralMessage" messageType="lns:referralMessage"/>
27     <variable name="referralMessage" type="xsd:boolean"/>
28  </variables>
29  <sequence name="main">
30     <receive name="processRFQ" partnerLink="buyerSellerRelation" portType="lns:rfqpw" operation="initiate" variable="rfqMessage"/>
31     <assign>
32      <copy>
33        <from opaque="yes"/>
34        <to variable="condition" property="xsd:boolean"/>
35      </copy>
36     </assign>
37     <switch name="quoteDecision">
38      <case condition="if bpws:getVariableData('condition') = true">
39       <invoke name="prepareReferral" partnerLink="buyerSellerRelation" portType="lns:rfqpwCallback" operation="onResult" inputVariable="quoteMessage"/>
40      </case>
41      <otherwise>
42       <invoke name="processQuote" partnerLink="buyerSellerRelation" portType="lns:rfqpwCallback" operation="onResult" inputVariable="referralMessage"/>
43      </otherwise>
44     </switch>
45  </sequence>
46 </process>
```

Listing 7: Abstract BPEL description

7 Conclusion

We have presented the multi meta-model process ontology (*m3po*), an intermediate unifying workflow ontology based on the most prominent existing workflow reference models. We have shown that *m3po* is unique in the combination of workflow primitives and support for choreography-specific concepts, and can act as connecting ontology to integrate workflow models and allow choreography interface extraction.

We have validated its completeness by first translating an example workflow from IBM Websphere MQ Workflow to *m3po* and then extracting a Abstract BPEL choreography interface from it. We showed that one can straightforwardly generate a complete and correct Abstract BPEL interface from the annotated *m3po* ontology.

To further verify the hypothesis that any process model can be mapped to the m3po, more work is still required on the construction of mappings from the workflow management systems and choreography languages to our ontology.

References

[1] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: Yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[2] 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.

[3] P. C. Attie, M. P. Singh, A. Sheth, and M. Rusinkiewicz. Specifying and enforcing intertask dependencies. In *Proc. of the 19th Conf. on Very Large Databases*, pp. 134–145. 1993.

[4] J. de Bruijn *et al.* The web service modelling language WSML. WSML Final Draft D16.1, 2005. V0.21.

[5] C. Bussler. *B2B Integration*. Springer, 2003.

[6] 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.

[7] 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.

[8] S. Damodaran and N. Kartha. Automating B2B integration with XML – the RosettaNet approach. *XML Journal*, 2004.

[9] R. Dijkman and M. Dumas. Service-oriented design: A multi-viewpoint approach. *Int. Journal of Cooperative Information Systems*, 13(4):337–368, Dec. 2004.

[10] 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.

[11] 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.

[12] M. Gruninger. Ontology of the process specification language. In S. Staab and R. Studer, (eds.) *Handbook on Ontologies*, pp. 575–592. Springer, 2004.

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

[14] N. Kavantzas *et al.* Web services choreography description language, Nov. 2005.

[15] A. Martens. Consistency between executable and abstract processes. In *Proc. of Intl. IEEE Conf. on e-Technology, e-Commerce, and e-Services*. Mar. 2005.

[16] S. Mehrotra, R. Rastogi, A. Silberschatz, and H. Korth. A transaction model for multidatabase systems. In *Proc. of the 12th Int. Conf. on Distributed Computing Systems*, pp. 56–63. 1992.

[17] M. zur Muehlen. Organizational management in workflow applications issues and perspectives. *Information Technology and Management*, 5(3-4):271–291, 2004.

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

[19] 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.

[20] 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.

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

[22] 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.

[23] A. P. Sheth, W. M. P. van der Aalst, and I. B. Arpinar. Processes driving the networked economy. *IEEE Concurrency*, 7(3):18–31, 1999.

[24] S. Thatte *et al.* Business process execution language for web services, v1.1, May 2003.

[25] XML process definition language v2.0. Workflow Management Coalition, 2005.