

m3po: An Ontology to Relate Choreographies to Workflow Models

Armin Haller and Eyal Oren
Digital Enterprise Research Institute
Galway, Ireland
firstname.lastname@deri.org

Paavo Kotinurmi
Helsinki University of Technology
Helsinki, Finland
paavo.kotinurmi@tkk.fi

Abstract

External business processes (choreography models) are currently disconnected from internal processes (workflow models), which leads to synchronisation and verification problems. Connecting these by 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 meta-models. We introduce our multi meta-model process ontology (m3po), which is based on various existing reference models and languages from the workflow and choreography domain. The m3po ontology relates workflow models to choreography models and allows choreography extraction from internal workflow models. An initial validation is given by translating an IBM Websphere MQ Workflow model into the m3po ontology and subsequently extracting an Abstract BPEL model from the ontology.

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 description and execution of business processes [9]. 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 or abstract processes, or provider behaviour [8, 24]), stating conversational patterns with which the services can be accessed. A choreography interface represents a subset of a choreography model, such that it describes the behaviour of one participant in a collaboration. A composition of at least two choreography interfaces constitutes a choreography model. A fundamental lack of descriptions of choreog-

raphy models in current frameworks such as the Business Process Execution Language (BPEL4WS) [24] and the Web Service Choreography Description Language (WS-CDL) [13], manifests the disconnection between the choreography model and the workflow model. Conceptually, a choreography interface of one partner can be regarded as an abstracted view on a workflow model (c.f. [5, 6, 8, 21]). However, current choreography frameworks ignore this dependency relation.

The disconnection between choreography interfaces and workflow models leads to two problems: choreography interfaces have to be manually synchronised with workflow models, and their consistency with regard to the workflow model can not be automatically verified. A major obstacle in connecting internal workflow models and external choreographies is the variety of existing workflow languages, workflow meta-models, and choreography languages [2, 19, 22]. Directly translating from workflow languages to choreography languages would require n^2 mappings, for each combination of workflow and choreography language.

2 Approach

Figure 1 describes our approach to connect workflow models and choreography models. We develop an intermediate unifying ontology, called *m3po* (multi meta-model process ontology). This ontology can be used to represent internal workflow models. Choreography interfaces corresponding to the internal workflow model can be extracted from the ontology, thus reducing the amount of mappings from a quadratic to a linear order of the number of models. The approach we follow to define a global schema is well known in data integration, whereas two ways to relate local schemes with a global schema have been proposed, namely global-as-view and local-as-view [14].

In this paper we focus on the definition of the ontology, which is based on a thorough analysis of existing workflow and choreography models. It represents a local-as-view approach, where the ontology is a stable

model which allows the mapping of different workflow models and further extraction of choreography models.

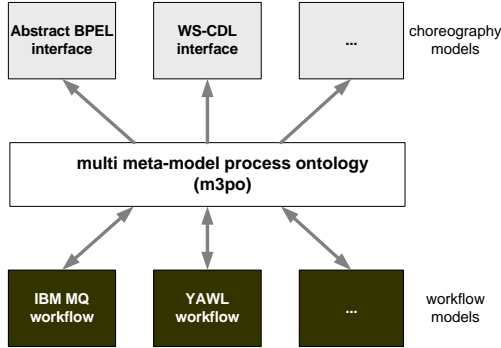


Figure 1: A global ontology to extract choreographies

2.1 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 [12] widely recognised as essential workflow characteristics: functional¹, behavioural, informational, organisational, and operational aspects. Finally, we compare the support for choreography-specific extensions, specifically the support for message definitions and message passing, and collaboration role-models.

2.2 Analysis

We include the following models in our analysis: the XML Process Definition Language (XPDL) [25], a standard language specifically designed to exchange workflow models; the Process Specification Language (PSL) [11], a formal ontology that can represent the semantics of workflow models and enables semantics-preserving exchange of models; Yet Another Workflow Language (YAWL) [1], a behaviourally complete workflow language with direct support for all workflow pat-

¹The functional aspect is not separately analysed.

terns [2]; BPEL4WS [24], the most prominent (executable) web orchestration language; and WS-CDL [13], a W3C candidate recommendation for a multi-party choreography language.

Table 1 shows a summary of the support of our evaluation criteria by the identified models and languages. A ‘+’ denotes full support of the aspect, ‘±’ marks aspects which are partly fulfilled (with an explanation of the limited support given below), and a ‘-’ indicates no support.

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

Table 1: Model support of evaluation criteria

Vertical integration is concerned with the integration of internal and external process models; it includes workflow views [6], abstraction levels, and visibility of processes and activities. In BPEL4WS a subset called Abstract BPEL allows the specification of choreography interfaces, but cannot indicate the visibility of process parts and is further disconnected from its counterpart, executable BPEL4WS. XPDL distinguishes between private and public processes, but they cannot be defined by the activity, nor can their visibility be parameterised for a participant. WS-CDL only describes choreographies and has no support for workflow modeling, whereas PSL and YAWL do not support choreography modelling.

Horizontal integration denotes the ability to deal with multiple workflow models. PSL offers a formal ontology that can express semantic differences between models and systems. XPDL offers workflow interoperability on a syntactic level, but cannot express semantic differences. YAWL, BPEL4WS, and WS-CDL do not address integration, but are stand-alone models.

In the *behavioural* aspect, YAWL supports all control-flow patterns [1]. XPDL, PSL, and WS-CDL support only the basic control-flow constructs (although PSL allows arbitrary extensions), whereas BPEL4WS supports also more advanced patterns [2]. Constraint-based approaches [3] are only supported by PSL. In the *informational* aspect, all investigated languages but PSL support data type definitions and data passing. PSL does not offer native relations to model data passing (although it can be added to the model) nor does it allow data typing. In the *organisational* as-

pect, XPDL supports the use of external resource definitions (relations between the resources can not be described). The other models in question do not include an organisational model. In the *operational* aspect, XPDL offers various invocation methods and styles. YAWL and BPEL4WS use Web service for invoking operations. PSL and WS-CDL are non-executable models and therefore do not cover this aspect. XPDL, BPEL4WS, and WS-CDL support *choreography specific* aspects, whereas PSL and YAWL do not.

3 Motivating Example

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

3.1 Current situation

An automotive parts vendor implements and executes his internal processes with IBM Websphere MQ Workflow². 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.

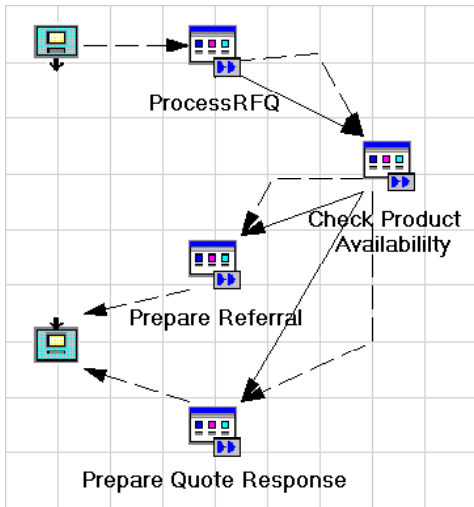


Figure 2: IBM MQ Workflow RFQ

The process starts with a RFQ from a customer. The vendor checks whether the requested part 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.

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

The graphic in figure 2 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 in the application within a property tab of an activity.

3.2 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 conform to industry standards this public process should conform to the standardised RosettaNet choreography interface PIP 3A1³; 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 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.

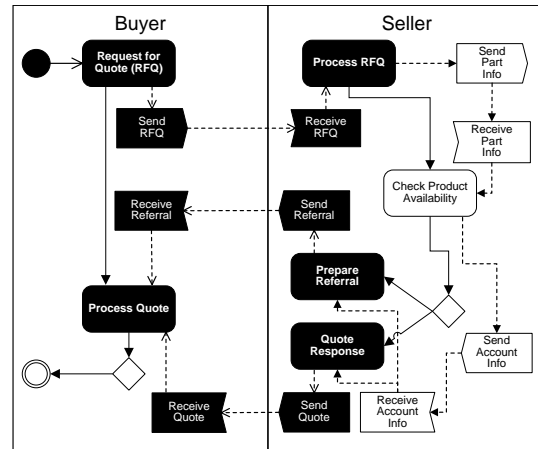


Figure 3: External process (RosettaNet PIP)

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

³<http://www.rosettanet.org/PIP3A1>.

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 [7].

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 WfMS.

4 Ontology

In the following sections we describe the multi meta-model process ontology *m3po*, introduced per workflow aspect. The ontology is written in the Web Service Modeling Language (WSML) [4], an ontology language for the web, making the semantics of the process concepts formal and explicit (which is necessary because of the different semantics of workflow meta-models). WSML is particularly suited for our requirements. First, identifiers (namespaces) allow the reuse of resources on the Web, which is important when data is modelled in a business process. Second, the frame-based modeling provides direct constructs for class membership, subclassing and attributes in the language.

For readability reasons, we display the important concepts in UML class diagrams⁴; 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 snippets⁵ from our example in section 3.

4.1 Functional and Behavioural Aspect

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

4.1.1 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 [9].

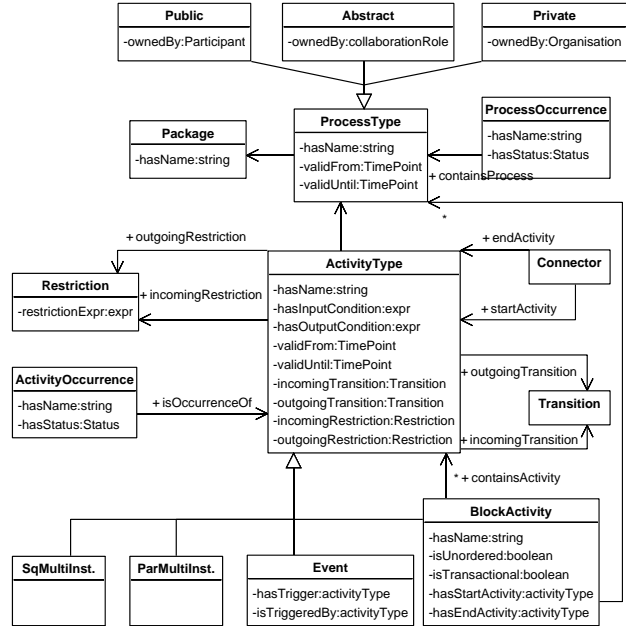


Figure 4: Functional and behavioural aspects

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 [12]. 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.

```

12 instance rfqpw memberOf publicProcessType
13   hasName hasValue "RFQ Processing Workflow"
14   hasPartnerLink hasValue buyerSellerRelation
49
50 instance rfq memberOf activityType
51   hasName hasValue "Process Request for Quote"
52   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 with well defined semantics, such as *active*, *suspended*, *resumed*, *can-*

⁴some ontology concepts are omitted from the diagrams

⁵the complete ontology instance for the example is available at <http://m3pe.org/ontologies/rfq.wsml>.

celled, 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 [9] 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. Listing 2 shows a snippet of an instance of our example model with explicit control-flow and data-flow dependency on the *request-for-quote* activity. *ParallelMultiInstantiation* and *sequentialMultiInstantiation* model structural patterns and patterns involving multiple instances. The *isUnOrdered* property on *blockActivities* models a non-deterministic ordering of tasks. For constraint-based modelling one can omit the *connectors* and define *input-* and *outputConditions* on the *activityTypes* instead.

```

58 instance cpa memberOf activityType
59   hasName hasValue "Check Product Availability"
60   hasSplitRestriction hasValue productAvailabilityCondition
61
62
63 instance rfqToCpa memberOf {controlConnector, dataConnector}
64   hasStartActivity hasValue rfq
65   hasEndActivity hasValue cpa

```

Listing 2: Example control- and data-flow connectors

4.1.2 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, *abstract-ProcessTypes* 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; 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 to the ontological model.

```

26 instance sourceNode memberOf {startEvent, messageEvent}
27   hasName hasValue "Source Node"
28   isVisibleFor hasValue buyer
29
30
31 instance checkStockApplication memberOf manualTask
32   hasName hasValue "Check Availability in Material Management"
33   hasPerformer hasValue warehouseman
34   isVisibleFor hasValue seller

```

Listing 3: Manual task with visibility assignments

4.2 Informational Aspect

The informational aspect (see figure 5) is defined by the data and data-flow perspectives [12]. 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.

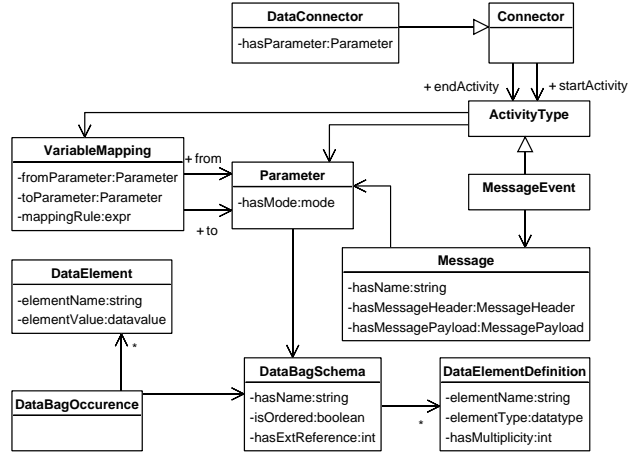


Figure 5: Informational aspect

4.2.1 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 datatypes⁶. The actual data values (i.e. *elementType*) passed in a process are defined on *activityOccurrences*.

```

37 instance processInput memberOf parameter
38   hasDataBagSchema hasValue partDescription
39   hasMode hasValue inParameter
40
41 instance partDescription memberOf dataBagSchema
42   hasName hasValue "Part Description"
43   hasDataElement hasValue rfqPartID

```

Listing 4: Example of explicit data passing

To model dataflow *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

⁶<http://www.w3.org/TR/xmlschema-2/>.

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.

4.2.2 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 workflow 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 *message-Header* (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*.

```

26 instance sourceNode memberOf {startEvent, messageEvent}
27   hasName hasValue "Source Node"
28   isVisibleFor hasValue buyer
29   hasMessage hasValue rfqMessage
30
31 instance rfq memberOf message
32   hasName hasValue "Request for Quote Message"
33   hasParameter hasValue processInput
34   hasMessageHeader hasValue rfqMessageHeader
35   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 the *fromRole* and *toRole*. A grounding to a specific partner interface is not necessary. The ontology further allows to define the visibility of *data-Bags* to a *collaborationRole*. This gives the modeler the opportunity to restrict the access to data elements.

4.3 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].

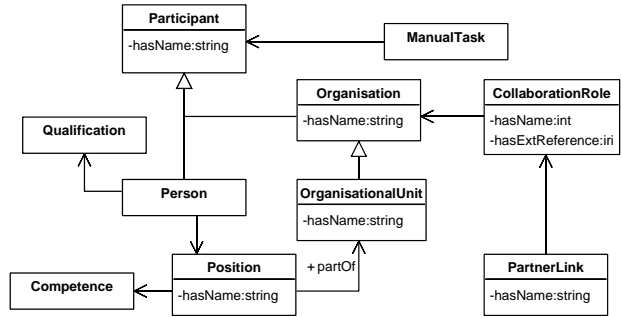


Figure 6: Organisational aspect

4.3.1 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*.

4.3.2 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 a choreography, 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.

```

16 instance buyerSellerRelation memberOf partnerLink
17   hasName hasValue "Buyer/Seller Relation"
18   hasRole hasValue {seller, buyer}
19
20 instance seller memberOf collaborationRole
21   hasName hasValue "Automotive Parts Vendor"
22
23 instance buyer memberOf collaborationRole
24   hasName hasValue "Car Manufacturer"

```

Listing 6: *PartnerLinks* and *collaborationRoles*

4.4 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 only models the execution properties.

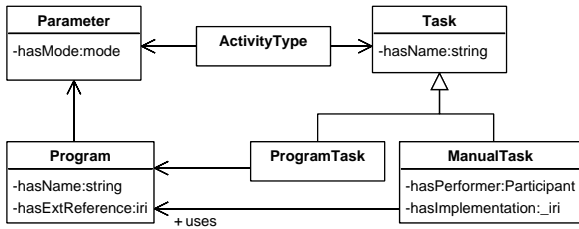


Figure 7: Operational aspect

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 4.2. *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.

4.5 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 [12] 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, retrievable, 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 Initial Validation

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.

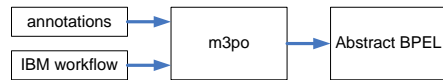


Figure 8: extracting a choreography interface

We developed XSLT scripts to map a common subset of the workflow patterns [2], which is expressible in both, BPEL4WS and IBM MQ Workflow, to and from our ontology. First, we mapped from the proprietary format used within MQ Workflow to *m3po*. Second, we manually annotated the model with choreography-specific properties supported by *m3po* and finally we applied a second XSLT script to map from *m3po* to a BPEL4WS description. We subsequently checked by executing the two models if the abstracted choreography model indeed behaves as the underlying workflow model.

The result of the mapping from the workflow model to *m3po* has been partly shown in the example snippets throughout section 4. A snippet of the extracted choreography interface in BPEL4WS is shown in listing 7⁷. The listing 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 listing was generated applying a further processing step, which removes all abstracted constructs. As such, 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" ><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="rfqpw" >
9 <partnerLink name="buyerSellerRelation" partnerLinkType="Ins:buyerSellerRelation"
10 myRole="seller" partnerRole="buyer" /></partnerLinks>
11 <variables>
12 <variable name="rfqMessage" messageType="Ins:rfq" />
13 <variable name="quoteMessage" messageType="Ins:quote" />
14 <variable name="referralMessage" messageType="Ins:referral" />
15 </variables>
16 <sequence name="main" >
17 <receive name="processRFQ" partnerLink="buyerSellerRelation"
18 portType="Ins: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="Ins:rfqpwCallback" operation="onResult"
26 inputValue="quoteMessage" />
27 </case >
28 <otherwise >
29 <invoke name="processQuote" partnerLink="buyerSellerRelation"
30 portType="Ins:rfqpwCallback" operation="onResult"
31 inputValue="referralMessage" />
32 </otherwise >
33 </switch >
34 </sequence >
35 </process >

```

Listing 7: Abstract BPEL description

6 Related Work

Additionally to the models we already evaluated in section 2 our work is most closely related to several approaches to views on process models [5, 6, 20, 21].

Chebbi *et al.* [5] 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.* [6] 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 and Orłowska [21] introduce a state-transition approach that binds states of private work-

⁷The PartnerLinkType definition is part of a WSDL file and is only aggregated in the listing for simplicity

flow 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 [12], Meteor [23], and CrossFlow [10]. These approaches require a pre-established partner agreement on the semantics of the process models and do not target current choreography standards.

Martens [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.

7 Conclusion and Future Work

We have presented the multi meta-model process ontology *m3po*, an intermediate unifying workflow ontology based on the most prominent existing workflow reference models and on choreography-specific constructs in the different choreography specification languages. Choreography-specific information is absent from current workflow models. Thus, *m3po* is unique in the combination of workflow primitives and choreography-specific concepts in one model.

m3po allows visibility definition of workflow elements to choreography roles. Such parameterised visibility definition is a key requirement to provide a sufficiently abstracted choreography model, since the externally visible behaviour should not include any private steps.

Although *m3po* is essentially a comprehensive meta model to define business processes, it is not intended for process modelling per se. Instead, it provides a global schema that relates internal and external processes and which can be used to extract a choreography model from an internal workflow model.

We have initially validated the completeness of *m3po*, but further validation is necessary. Given the different representational power of the target choreography languages, complete mappings from arbitrary workflow models cannot be guaranteed: although the ontology allows definition extensions to cover newly encountered constructs, these are not necessarily expressible in the target choreography language. Future work will investigate which parts of workflow models can be mapped to which choreography languages.

Acknowledgement

This material is based upon works supported by the Science Foundation Ireland under Grant No. SFI/04/BR/CS0694, the Finnish Funding Agency for Technology and Innovation (Tekes) and the Graduate School for Electronic Business and Software Industry.

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] 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.
- [6] 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. *Inf. Tech. and Management*, 5(3-4):221–250, 2004.
- [7] S. Damodaran and N. Kartha. Automating B2B integration with XML – the RosettaNet approach. *XML Journal*, 2004.
- [8] R. Dijkman and M. Dumas. Service-oriented design: A multi-viewpoint approach. *Int. Journal of Cooperative Information Systems*, 13(4):337–368, 2004.
- [9] 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.
- [10] P. Grefen, K. Aberer, H. Ludwig, and Y. Hoffner. Crossflow: Cross-organizational workflow management for service outsourcing in dynamic virtual enterprises. *IEEE Data Eng. Bulletin*, 24(1):52–57, 2001.
- [11] M. Gruninger. Ontology of the process specification language. In S. Staab and R. Studer, (eds.) *Handbook on Ontologies*, pp. 575–592. Springer, 2004.
- [12] S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation*. Int. Thomson Computer Press, 1996.
- [13] N. Kavantzias *et al.* Web services choreography description language. 2005.
- [14] M. Lenzerini. Data integration: a theoretical perspective. In *Proc. of PODS*, pp. 233–246. 2002.
- [15] A. Martens. Consistency between executable and abstract processes. In *Proc. of Intl. IEEE Conf. on e-Technology, e-Commerce, and e-Services*. 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. M. 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. Orłowska. Facilitating cross-organisational workflows with a workflow view approach. *Data Knowl. Eng.*, 51(1):109–147, 2004.
- [22] A. Sheth, W. M. P. van der Aalst, and I. B. Arpinar. Processes driving the networked economy. *IEEE Concurrency*, 7(3):18–31, 1999.
- [23] A. Sheth, *et al.* The METEOR workflow management system and its use in prototyping significant healthcare applications. In *Proc. of TEPR*, pp. 267–278. 1997.
- [24] S. Thatte *et al.* Business process execution language for web services, v1.1. 2003.
- [25] WFMC. XML process definition language v2.0. Workflow Management Coalition, 2005.