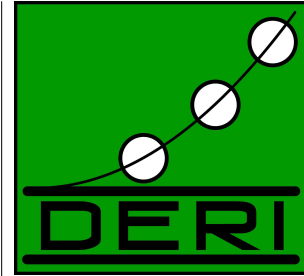


DERI – DIGITAL ENTERPRISE RESEARCH INSTITUTE



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 – DIGITAL ENTERPRISE RESEARCH INSTITUTE

**DERI Galway**  
University Road  
Galway  
IRELAND  
[www.deri.ie](http://www.deri.ie)

**DERI Innsbruck**  
Technikerstrasse 13  
A-6020 Innsbruck  
AUSTRIA  
[www.deri.ie](http://www.deri.ie)



## 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<sup>1</sup>

Eyal Oren<sup>1</sup>

**Abstract.** Cross-organisational interoperability is a key issue for success in B2B e-commerce applications. To achieve this interoperability, choreography descriptions are necessary that describe how the business partners can cooperate. In existing approaches, these choreography descriptions are disconnected from the internal workflows of the partners.

We present an ontology that unifies both internal and external business processes, based on various existing reference models and languages from the workflow and choreography domain. The interoperability problems in this domain require an intermediate ontology to reduce the amount of necessary mappings.

Our ontology allows the extraction of choreography interface descriptions from internal workflow models. We demonstrate the ontology by translating an IBM Workflow MQ model into Abstract BPEL.

---

<sup>1</sup>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

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Motivating Example</b>	<b>2</b>
<b>3</b>	<b>Background</b>	<b>3</b>
<b>4</b>	<b>Related Work</b>	<b>3</b>
<b>5</b>	<b>Ontology</b>	<b>5</b>
5.1	Functional and Behavioural Aspect . . . . .	5
5.1.1	Choreography related properties . . . . .	7
5.2	Informational Aspect . . . . .	8
5.2.1	Choreography related properties . . . . .	9
5.3	Organisational Aspect . . . . .	10
5.3.1	Choreography related properties . . . . .	11
5.4	Operational Aspect . . . . .	11
5.5	Orthogonal Aspects . . . . .	12
<b>6</b>	<b>Validation</b>	<b>12</b>
<b>7</b>	<b>Conclusion</b>	<b>13</b>

## 1 Introduction

Organisations have long used process modelling to describe the dynamic behaviour of their business. Workflow Management Systems are commonly applied for process modelling and allow to describe and execute business processes [9]. With the advent of Service Oriented Computing [17] organisations started to expose their business functionality explicitly as reusable and composable services. For using these services organisations offer choreography interfaces (also called abstract processes or interface behaviour), stating conversational patterns in which the services can be consumed. These choreography interfaces are used when partners enter into a cross-organisational collaboration; they capture the interactions of the participating organisations and their dependencies.

A fundamental lack in current approaches such as Abstract BPEL [22] and WS-CDL [13], is the disconnection between the external choreography interfaces and the internal workflow descriptions. Conceptually, a choreography interface is an abstracted view on a business process, which is described by a workflow. Current approaches for choreography descriptions do not recognise this dependency. The disconnection between choreography interfaces and workflow definitions leads to two problems: choreography interfaces have to be manually synchronised with workflow descriptions, and it is not possible to automatically verify consistency of internal workflow descriptions and external choreography interfaces.

Figure 1 describes our approach to connect workflows and choreography descriptions; choreographies are public abstractions of internal workflows and can be semi-automatically extracted from workflows. Since a direct mapping from workflow languages to choreography languages would require  $n^2$  mappings (for each combination of workflow and choreography language), we have developed an intermediate unifying workflow ontology, thus needing only  $2n$  mappings.

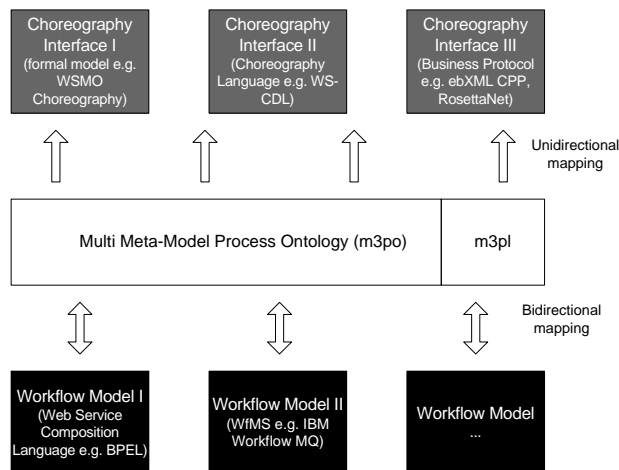


Figure 1: Connecting workflow models to extract choreography interfaces

This ontology is based on previous work on workflow interoperability. Several approaches exist for resolving workflow interoperability, most notable the workflow exchange language XPDL [23] and the Process Specification Language (PSL) [11]. These approaches are a good starting point because they unify different workflow metamodels in their aim for workflow interoperability, although they do not consider the choreography domain.

Our ontology integrates the workflow and choreography domains and enables the extraction

of choreography interfaces from workflow models. The ontology is not bound to specific workflow management systems or specific choreography languages; it incorporates both activity-based and constraint-based workflows, and captures all control-flow [2] and data-flow patterns [18].

## 2 Motivating Example

We illustrate the problems that companies face when designing collaborative business processes with an example.

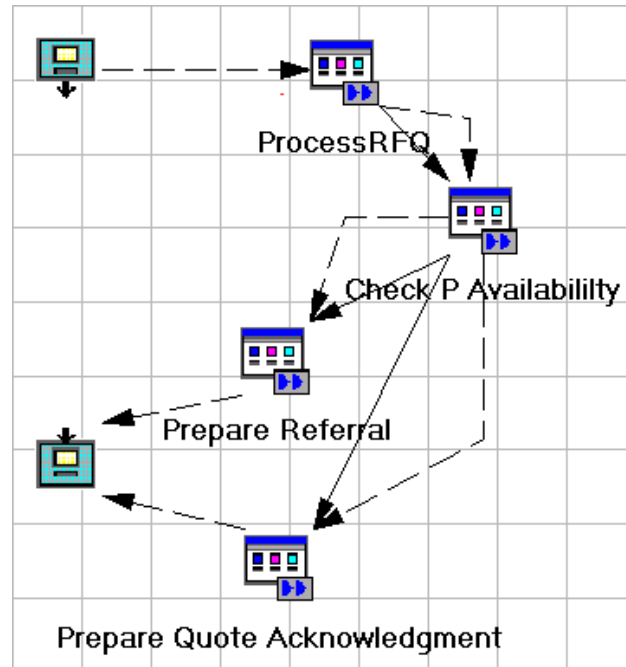


Figure 2: IBM Workflow MQ Request for Quote

An automotive parts vendor implements his internal process with IBM Workflow MQ. One part of his workflow concerns requests for quotes. Figure 2 shows a simplified view of the current processing of the purchase order request. The symbols on the left of figure 2 denote a source and sink node and represent the start and end of the IBM Workflow MQ process model. Dashed arrows show data transferred between activities and solid arrows denote the control flow. The process starts with a request for quote data input. The vendor prepares a quote if the product is available, otherwise he returns a referral to another vendor.

Now the vendor wants to conform his workflow to the standardised RosettaNet<sup>1</sup> PIP 3A1, that exactly describes such a request-for-quotation. Figure 3 shows this RosettaNet collaboration in a UML activity diagram, with public activities displayed in black and private activities in white. The supplier's choreography is formed by the black activities in his swimlane. The automotive vendor must therefore conform his internal workflow from figure 2 to this choreography interface so that his business partners can collaborate with him.

<sup>1</sup><http://www.rosettanet.org>.



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

Table 1: workflow and choreography features

XPDL [23] is a standard for exchanging workflow models, without runtime information between different workflow management systems. PSL [11] is an ontology that allows one to capture the semantics of workflow models and enables translations of models between workflow management systems. YAWL [1] is a research workflow language that supports all workflow patterns directly. BPEL [22] is an executable business process language and includes an abstract protocol. WS-CDL [13] is a multi-party collaboration model.

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

*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. XPDL has private and public processes, but they cannot be defined on the activity level, nor can the visibility be parameterised for a participant. Abstract BPEL describes choreographies but cannot indicate visibility, and is disconnected from executable BPEL. WS-CDL only describes choreographies and has no notion of internal processes.

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

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

In the *informational* aspect, XPDL, YAWL, BPEL, and WS-CDL support data type definitions and direct 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 definitions 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 are not executable and do not cover this aspect.

On *choreography specific* aspects, in particular support for message definition and passing and collaboration role-models is important. XPDL, BPEL, and WS-CDL support this, whereas PSL and YAWL do not cover choreography aspects.

## 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.

Our ontology is organised according to the five key aspects of workflow specifications and workflow management [12]. A short description of each aspect is given in the respective paragraphs.

Existing workflow models put a different emphasis on these five aspects. Some focus on the behavioural ordering of tasks, others allow a time centred scheduling, and again others concentrate on the representation of the organisational structure of a company.

In this research, we have identified the most elaborate reference model for each aspect and combined these models into m3po. We then extended the ontology with additional information specific to external process models such as message transfer, a collaboration role model, and the direction of communication.

We have then axiomatised the semantics of the constructs in order to facilitate correct and complete exchange of process information, so that a workflow engineer can create valid mappings between different workflow models.

The ontology can represent different modelling paradigms such as event-based, constraint-based and communication-based models [3, 9]. We have defined an operational semantics for the modelling primitives, but we do not restrict models to follow only one paradigm. It is therefore possible that a valid workflow (in the semantical sense) is modelled, but which is not a sensible model (in the workflow domain).

Our ontology is fully extensible: if a construct is lacking, the term can be added to the ontology and its semantics can be given by adding axioms to the knowledge base. The ontology is written in WSMML [4], a formal web ontology language; it can model the workflow and choreography domain and gives formal semantics to the ontology.

For readability reasons, we display the most important concepts in each aspect in a UML class diagram<sup>3</sup>. In the following sections we describe the ontology and we illustrate it with example snippets<sup>4</sup> representing the internal process from section 2. The ontology also captures choreography-related concepts that are not part of the internal workflow description: instances of these additional concepts would have been added to the WSMML representation of the IBM Workflow MQ model by a subject matter expert.

### 5.1 Functional and Behavioural Aspect

The functional and behavioural aspects of the ontology are shown in figure 4. The most important concepts are *processType*, *processOccurrence*, *activityType* and *activityOccurrence*.

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 constraints the ordering of other *activityTypes* or a task with pre- and postconditions to model constraint-based workflow specification languages [9].

```

12 instance rfqpw memberOf publicProcessType
13   hasName hasValue "RFQ Processing Workflow"
14   hasPartnerLink hasValue buyerSellerRelation

```

<sup>3</sup>the class diagram is intended as a guide to the ontology, rather than as its replacement; the full ontology can be found at <http://m3pe.org/ontologies/m3po.wsml>.

<sup>4</sup>the complete ontology for that example is available at <http://m3pe.org/ontologies/rfq.wsml>.

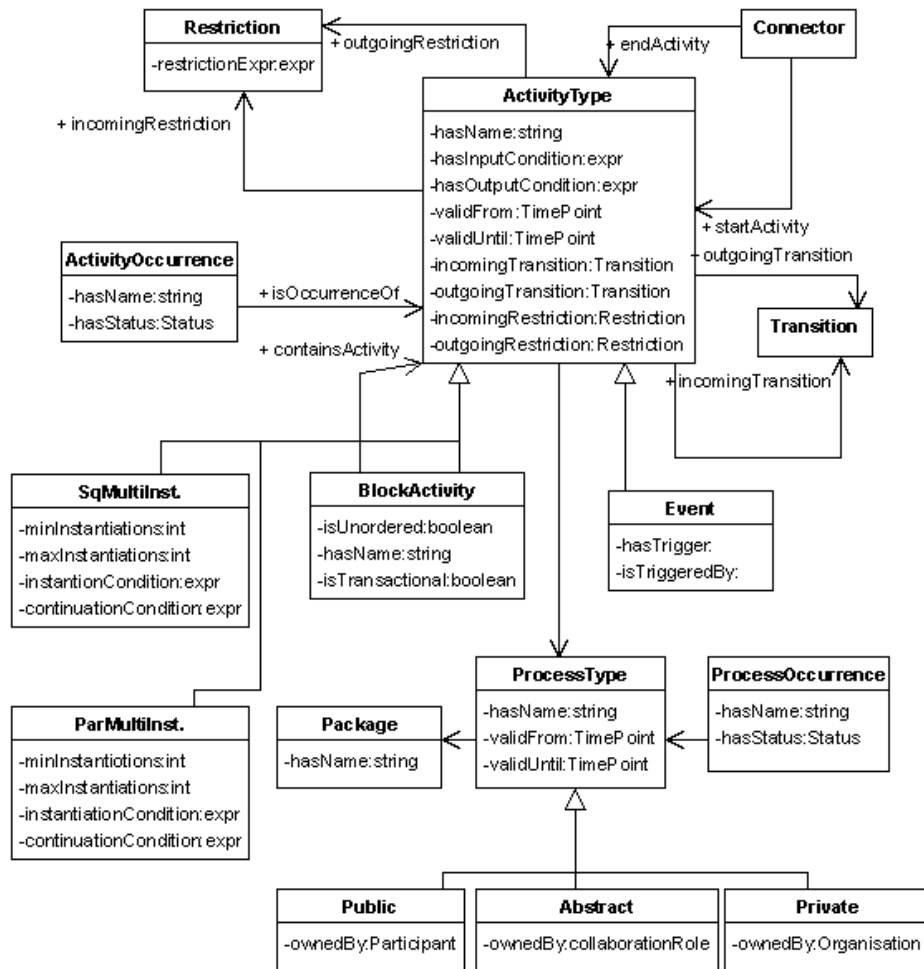


Figure 4: Functional and behavioural aspects

```

49
50 instance rfq memberOf activityType
51   hasName hasValue "Process Request for Quote"
52   hasTask hasValue saveDataInDatabase

```

Listing 1: Example use of *processType* and *activityType*

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 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 workflow management systems), such as *active*, *suspended*, *resumed*, *cancelled*, *aborted*, or *completed* (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. 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.

```

58 instance cpa memberOf activityType
59   hasName hasValue "Check Product Availability"
60   hasSplitRestriction hasValue productAvailability
61
62
63
64
65
66
67
68
69 instance rfqToCpa memberOf {controlConnector, dataConnector}
70   hasStartActivity hasValue rfq
71   hasEndActivity hasValue cpa

```

Listing 2: Example use of *controlConnector*

### 5.1.1 Choreography related properties

To extract choreographies from internal business processes, the ontology 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 (which implies a visibility to the owner of the process, the seller), whereas the *manualTask* is only visible to the seller (process owner).

```

26 instance sourceNode memberOf {startEvent, messageEvent}
27   hasName hasValue "Source Node"
28   isVisibleFor hasValue buyer
62
63 instance checkStockApplication memberOf manualTask
64   hasName hasValue "Check Availability in Material Management"
65   hasPerformer hasValue warehouseman
67   isVisibleFor hasValue seller

```

Listing 3: Example use of *manualTask* including visibility assignments

## 5.2 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.

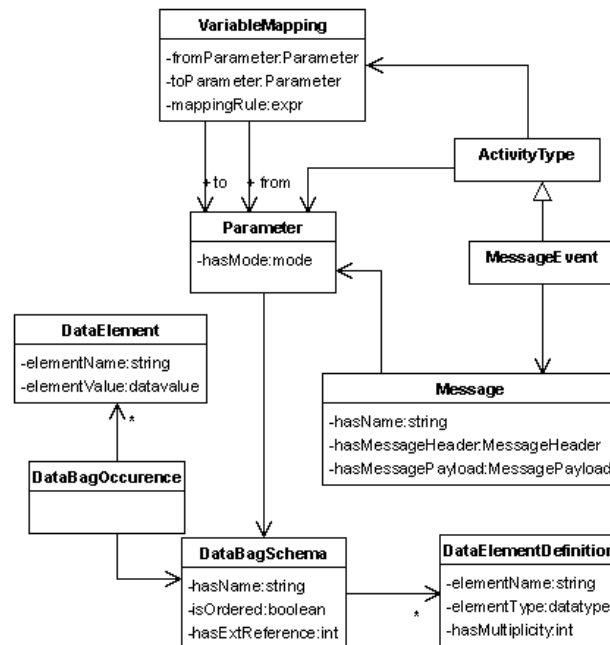


Figure 5: Informational aspect

The ontology supports all common data patterns [18]: 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 and hold 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<sup>5</sup>.

```

26 instance sourceNode memberOf {startEvent, messageEvent}
29   hasMessage hasValue rfqMessage
30
31 instance rfqMessage memberOf message
32   hasName hasValue "Request for Quote Message"
33   hasParameter hasValue processInput
34   hasMessageHeader hasValue rfqMessageHeader
35   hasMessagePayload hasValue rfqMessagePayload
36
37 instance processInput memberOf parameter
38   hasDataBagScheme hasValue partDescription
39   hasMode hasValue inParameter
40
41 instance partDescription memberOf dataBagScheme
42   hasName hasValue "Part Description"
43   hasDataElement hasValue rfqPartID

```

Listing 4: Example use of *dataBagSchemas* including *messages*

To model dataflow between activities, processes, and programs, *processTypes*, *activityTypes*, *programs* and *dataBags* can specify *parameters*. These parameters can be in-only (defining the input data), out-only (defining the output data or return value), or in/out (defining input data which is being modified). Listing 4 shows another part of our motivating example using *parameters* to pass *dataBagSchemas*.

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 *processTypes* include a *parameter* attribute. 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.

### 5.2.1 Choreography related properties

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 also shows the modelling of messages, which represents an annotation added to the internal process by a subject matter expert.

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*.

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.

<sup>5</sup><http://www.w3.org/TR/xmlschema-2/>.

### 5.3 Organisational Aspect

The organisational aspect (see figure 6) defines who is responsible for carrying out a specific *task*. The organisational elements (such as the hierarchical structure, organisational units, project teams and employees) have to be present and made accessible in some way. This model is typically kept separate from the process model, because managing resources inside the workflow management system may lead to problems if different systems are used for the implementation of a complex process [8].

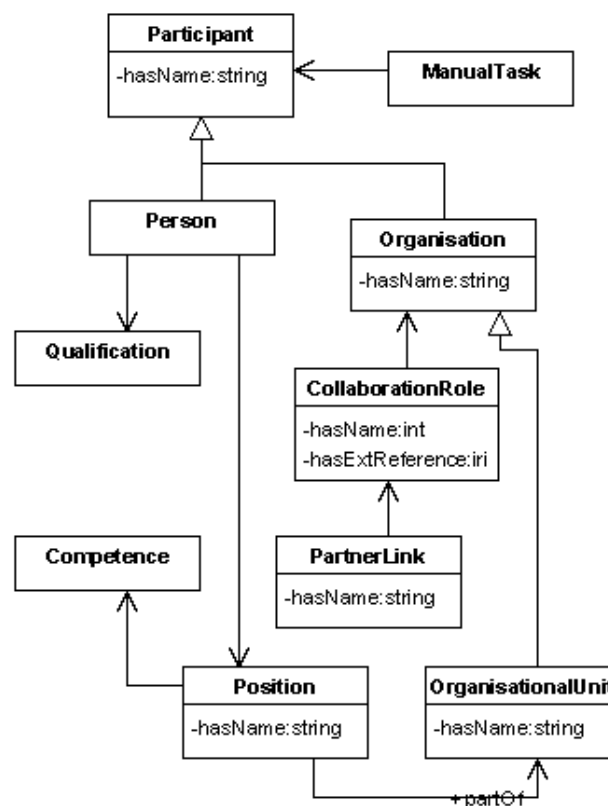


Figure 6: Organisational aspect

Since some workflow management systems do provide rudimentary resource management we include an adapted form of the organisational reference model introduced in [16].

A *participant* represents a organisational or human process resource. It is a named privilege granted to a *programTask* or *manualTask*. The recursive *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*.

### 5.3.1 Choreography related properties

The modelling of choreographies requires an additional role model different to the internal role model. 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 (see listing 5, which shows the role model from our example).

```

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 5: Example use of *partnerLinks* and associated *collaborationRoles*

### 5.4 Operational Aspect

The operational aspect of the ontology is shown in figure 7. Current workflow management systems 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: as (language-neutral) web services, as EJB applications, as an executable script, etc.

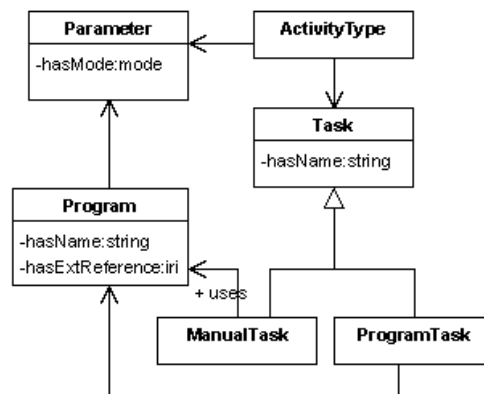


Figure 7: Operational aspect

To represent this, operations are implemented by a *manualTask* or a *programTask*. The operational aspect can be seen as an interface of the process model to external application, whereas the ontology models the following properties to define the execution:

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

Time is an important modelling concept, particularly in scheduling applications, which enables the representation of scheduling problems as a collection of scheduling constraints, such as activity durations, release dates, due dates, resource availability etc. These constraints in turn are used to provide schedules in which activities are assigned to resources over different time intervals.

Several workflow management systems allow rudimentary scheduling based on time. The ontology 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 *timeTriggerEvents* 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 [15]. 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.

Security in workflow management systems is handled by the access restrictions imposed by the organisational aspect [12]. The access control on the protocol level should therefore coincide with the organisational policies restricting the access to resources. Security related issues are therefore not in the scope of our ontology.

## 6 Validation

In the previous section we have outlined the modelling constructs m3po offers to allow one to specify mappings from the various internal process models. To define a generic mapping algorithm a workflow engineer has to first identify and clearly define the concepts intrinsic to each workflow management system. He has to write a mapping algorithm which translates concepts within the system or language to concepts within m3po. As mentioned above, once a workflow management system becomes “m3po compliant” (i.e. once a proven translator is written to/from m3po), it is possible to add choreography related information to the model and ultimately extract arbitrary

any choreography language.

Given the complexity of workflow management systems we have not completed a generic mapping algorithm yet. Instead, we can present our initial results, namely mappings from IBM Workflow MQ and to Abstract BPEL. We validated the model annotated with choreography related constructs by mapping it to an Abstract BPEL process.

We have already shown snippets of the ontological representation of the example process (of the automotive vendor) in section 5. Figure 8 presents the BPEL diagram that can be extracted from the ontology<sup>6</sup>.

Further empirical research on the mapping of different workflow models is necessary to validate the completeness of our ontology.

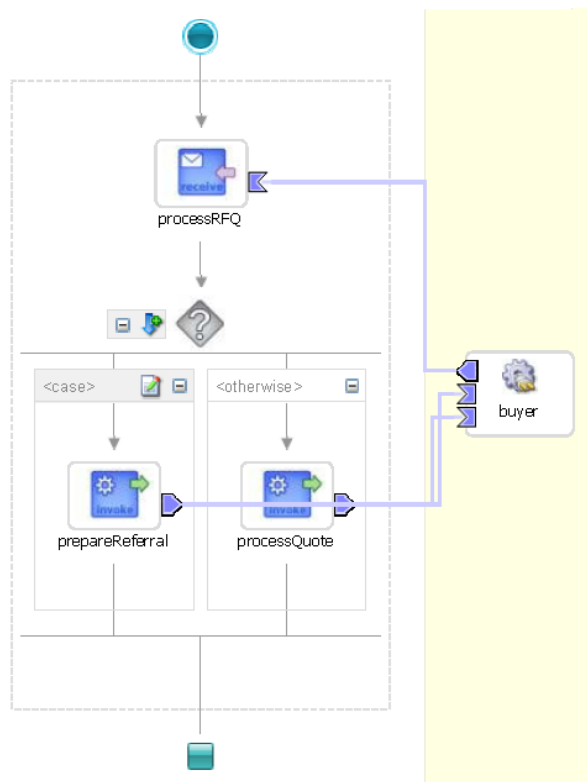


Figure 8: Choreography interface in Abstract BPEL

## 7 Conclusion

In existing approaches in the workflow and choreography domain, the choreography descriptions are independent of the internal workflows of the partners.

We have presented m3po, which is based on various reference models in the workflow and choreography domain and combines their modelling features. The m3po can act as a connecting ontology to integrate different workflow models. It has the unique feature of extracting external processes from an internal workflow model. We have shown how the ontology can be used to extract

<sup>6</sup>the full BPEL code for this process can be found at <http://m3pe.org/ontologies/rfq.bpel>.

such a choreography interface of an example workflow in a RosettaNet collaboration. We do not claim yet that our ontology completely supports arbitrary any horizontal integration, as we have not yet completed the mappings (and their validations) from different existing systems.

We have performed an initial validation on a workflow model (i.e. IBM Workflow MQ) and a choreography language (i.e. Abstract BPEL) respectively. More work is required to verify the completeness of our ontology with more mappings from other workflow systems, and more mappings to choreography languages.

## 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 *Proceedings of the 19th Conference 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] S. Ceri, P. Grefen, and G. Sanchez. Wide: A distributed architecture for workflow management. In *Proceedings of RIDE*. Birmingham, UK, 1997.
- [6] 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.
- [7] R. Dijkman and M. Dumas. Service-oriented design: A multi-viewpoint approach. *International Journal of Cooperative Information Systems*, 13(4):337–368, Dec. 2004.
- [8] W. Du and M. Shan. Enterprise workflow resource management. In *Proc. of the 9th Int. Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises*, pp. 108–115. 1999.
- [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. W. P. J. 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.
- [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*. International Thomson Computer Press, 1996.
- [13] N. Kavantzias *et al.* Web services choreography description language, Nov. 2005.
- [14] A. Martens. Consistency between executable and abstract processes. In *Proceedings of Intl. IEEE Conference on e-Technology, e-Commerce, and e-Services (EEE'05)*. IEEE Computer Society Press, Mar. 2005.
- [15] S. Mehrotra, R. Rastogi, A. Silberschatz, and H. Korth. A transaction model for multidatabase systems. In *Proceedings of the 12th International Conference on Distributed Computing Systems*, pp. 56–63. 1992.
- [16] M. zur Muehlen. Organizational management in workflow applications issues and perspectives. *Information Technology and Management*, 5(3-4):271–291, 2004.
- [17] M. P. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Communications of the ACM*,

46(10):25–28, 2003.

- [18] 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.
- [19] K. A. Schulz and M. E. Orłowska. Facilitating cross-organisational workflows with a workflow view approach. *Data Knowl. Eng.*, 51(1):109–147, 2004.
- [20] A. Sheth, *et al.* The meteor workflow management system and its use in prototyping significant healthcare applications. In *Proceedings of the 1997 Toward an Electronic Patient Record Conference (TEPR'97)*, pp. 267–278. Nashville, TN, USA, 1997.
- [21] 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.
- [22] S. Thatte *et al.* Business process execution language for web services, v1.1, May 2003.
- [23] XML process definition language v2.0. Workflow Management Coalition, 2005.