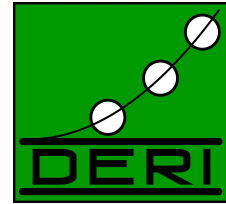


DERI – DIGITAL ENTERPRISE RESEARCH INSTITUTE



## A PROCESS ONTOLOGY FOR BUSINESS INTELLIGENCE

Armin Haller      Mateusz Marmolowski  
Eyal Oren        Walid Gaaloul

DERI TECHNICAL REPORT 2008-04-01  
APRIL 2008

DERI – DIGITAL ENTERPRISE RESEARCH INSTITUTE

**DERI Galway**  
IDA Business Park  
Galway, Ireland  
[www.deri.ie](http://www.deri.ie)



DERI TECHNICAL REPORT  
DERI TECHNICAL REPORT 2008-04-01, APRIL 2008

A PROCESS ONTOLOGY FOR BUSINESS INTELLIGENCE

Armin Haller<sup>1</sup>, Mateusz Marmolowski<sup>1</sup>, Eyal Oren<sup>2</sup>, Walid Gaaloul<sup>1</sup>

**Abstract.** This paper presents oXPDL, a process interchange ontology based on the standardised XML Process Definition Language (XPDL). XPDL was introduced to allow process model exchange between information systems, most of which are based on proprietary workflow models. In its current form, XPDL allows only syntactic vendor-specific extensions without clear semantics, has only limited support for informational and organisational modelling aspects and cannot be interlinked to existing standardised knowledge bases. Our process interchange ontology oXPDL explicitly models the complete semantics of XPDL process models in a standard Web ontology language. The oXPDL ontology has a strong focus on the reuse and integration of existing standard ontologies such as PSL, RosettaNet, SUMO and eClassOWL. We present the ontology and the accompanying tool to automatically translate an XPDL process model to its corresponding oXPDL. The oXPDL process models may be used for integrated process analysis, by querying and reasoning over multiple models, each of which may originate from different information systems, in combination with business rules described in background ontologies.

---

<sup>1</sup>Digital Enterprise Research Institute (DERI), National University of Ireland, Galway, IDA Business Park, Lower Dangan, Galway, Ireland. E-mail: {armin.haller, mateusz.marmolowski, walid.gaaloul}@deri.org.  
<sup>2</sup>AI department, VU Amsterdam, de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands. E-mail: eoren@few.vu.nl

**Acknowledgements:** This material is based upon works supported by the Science Foundation Ireland under Grant No. SFI/02/CE1/I131 and No. SFI/04/BR/CS0694.

Copyright © 2008 by the authors

## 1 Introduction

Improving the efficiency of business processes is a major driver of information technology in current businesses. The majority of enterprise information systems are process-aware, using some mechanism to support, control, and monitor business processes [5]. Typical examples of such systems, driven by implicit or explicit process models, are *enterprise resource planning* systems (ERP), *customer relationship management* systems (CRM), *groupware* collaboration systems and *workflow management* systems (WfMS), specifically built for modelling and executing business processes.

All these tools operate on workflow models, typically focused on control-flow captured in procedural process notations. Some more standardised modelling languages exist, such as the Business Process Execution Language (BPEL) [23], the Business Process Modelling Notation (BPMN) [27] and Event-Driven Process Chains (EPCs). On the other hand, proprietary representations are widely used in commercial systems, such as the FlowMark Definition Language<sup>1</sup>, used in the IBM WebSphere MQ Workflow system<sup>2</sup>. Despite partial standardisation of process models, no consensus exists on the representation or conceptual model of complete workflows [22], hampering process interoperability and exchange.

### 1.1 The XPDL process format

The XML Process Definition Language (XPDL) [25] has been developed for describing and exchanging process models between different applications and their proprietary models, and is currently supported by over seventy products<sup>3</sup>. XPDL is a non-executable description language, represented as an XML Schema definition of workflow concepts and properties, with several predefined extension points. With version 2.0 it is designed as a standard serialisation format of BPMN. In contrast to BPEL, which can be seen as the quasi-standard for process execution over Web services, it serves as a description language only and is not meant for execution.

In this paper, we focus on a business intelligence scenario, where analysis data is gathered from several process-aware systems within a single enterprise. Since process data is combined from different systems, the data needs to be integrated before the business analysis can be performed. Furthermore, the process data should typically be combined with business information such as partner data, stock data, or other background knowledge. In terms of interoperability and integration with other sources of data, especially in business intelligence operations to identify potential re-engineering situations, XPDL poses several challenges:

- XPDL lacks support for vertical (public-private) integration concepts such as workflow views, partner roles, or parametrised activity visibility [11].
- The semantics of the standard XPDL elements are defined only informally, leading to interpretation differences between vendors [9].
- Querying XPDL models with XPath and XQuery ignores process and data semantics and does not support integrated inspection of multiple models.

---

<sup>1</sup>Export/import facilities for BPEL are provided for the behavioural aspect.

<sup>2</sup><http://www-306.ibm.com/software/integration/wmqwf/>

<sup>3</sup><http://www.wfmc.org/standards/xpdl.htm>

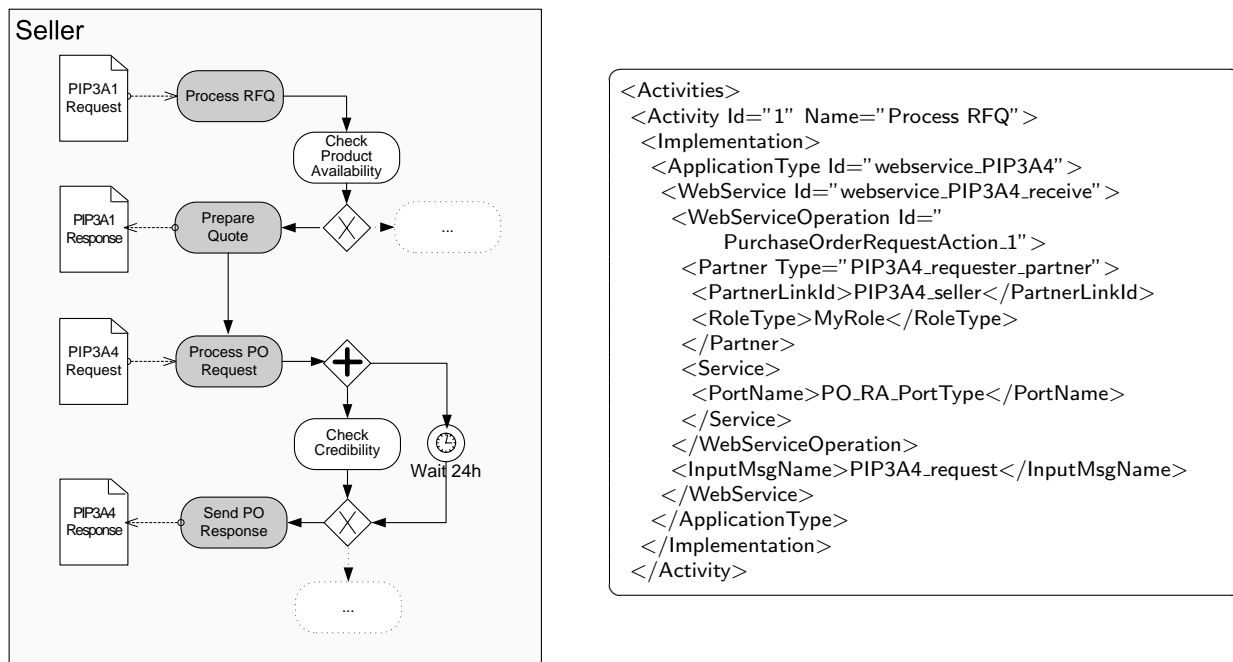


Figure 1: RosettaNet quoting and purchasing process (BPMN)

- XPD L elements, developed as most general common denominators between existing process languages, abstract from much relevant information. For example, data concept hierarchies are not represented in XPD L.
- XPD L extensions are limited to XML Schema definitions. Since these support only few semantic constraints [9], extensions cannot be interpreted or used for process analysis without prior knowledge of the particular extension and cannot form the basis for further extensions.

## 1.2 Motivating example

We illustrate these challenges with a simple XPD L process of a quoting and purchasing scenario, defined in the RosettaNet standards PIP3A1 and PIP3A4. The BPMN representation of the process is shown on the left of Figure 1, an excerpt of the corresponding XPD L document is shown on the right. The white-coloured activity boxes denote parts of the internal computational steps, dotted boxes are placeholders for omitted internal computation steps; the dark-coloured boxes represent the public behaviour according to PIP3A1 and PIP3A4. The arrows in BPMN constrain the behavioural ordering of activities, the diamond-shaped boxes denote routing decisions (‘+’ means a parallel split, ‘×’ a conditional split).

In a business analysis scenario, we would like to answer a defined set of competency questions [26] based on the process model and background ontologies such as an ontologised version of RosettaNet [9] or eClassOWL [12] among others. The semantics of the terms used within the following competency questions are used in compliance to their definition in the XPD L specification.

1. In which processes do all *members* of an organisation participate?
2. Which activities belong to a certain *pool*?
3. Which activities of a particular process belong to some *subprocess*?
4. Which *events* can be triggered in a particular process model?
5. What message types are received by a *pool* in a particular package?
6. Does a *message* include some substitutable product?
7. In which processes does a certain *actor* participate in?
8. Which partners should implement a web service *operation* in a process type?
9. Which tasks precede “prepare quote” in instances of the “PIP3A1” process?
10. What is the most frequent path in instances of the “PIP3A1” process?

To answer these questions correctly, we need to integrate information from multiple sources and process-aware systems, while still respecting the process semantics of these models. Existing process-aware systems such as WfMS do not offer standardised APIs for querying and combining their data and do not support data integration with other process-aware systems; consequently, answering these competency questions requires additional management information systems for data integration and analysis, and requires the manual construction of mapping and transformation rules to convert and align the data from various systems.

### 1.3 Approach: ontologising and interlinking XPDL

To address the interoperability challenges that arise when answering these competency questions, we explicitly represent the informal underlying semantics of XPDL in a formal and expressive Web ontology language. We do not impose our interpretation on process models, but carefully formalise the semantics specified in the XPDL standard. Formalising the XPDL standard allows automatic translation of XPDL models (instance documents) into Web documents based on standardised Web ontology languages such as OWL and WSML. Based on these Web ontology languages, the ontologised process models can be interlinked and enriched with business knowledge in existing background ontologies. Using Web ontology languages to represent process models has several benefits:

- Explicitly representing the complete semantics of process models (including their behavioural semantics and the semantics of the data) improves the interoperability between partners since implementers have direct access to the semantics of informational and organisational elements and non-default process elements.
- Ontologically representing process models allows querying on a relatively high level of abstraction. In contrast to traditional approaches, the queries on models and instances are not limited to specific vendors, but can be populated from multiple process-aware systems that support XPDL export (which is available in all leading products). Ontological representations simplify model integration, allowing business intelligence operations over multiple instances of the same or different process models.

- The use of Web ontology languages enables interlinking and relating the process model to existing business dictionaries and background knowledge such as the supply-chain information in ontologised RosettaNet [9] or the catalogue information in eClassOWL [12]. Reusing such standardised background ontologies reduces the modelling effort in the informational and organisational aspects and helps business analysts to formulate queries that range over multiple models.
- Since the same Web ontology languages are used in semantic Web Service models such as WSMO [21] and OWL-S [15], ontological representations of process models support consistency between external service descriptions (choreography) and internal process models (workflow) to ultimately generate choreography interfaces from process models [11].

In this paper, we present an approach for representing XPDL processes semantically, by translating them to our oXPDL ontology that captures the implicit and explicit semantics of that process model. We follow a well-known methodological ontology engineering approach [26] based on the presented competency questions which describe additional requirements on the ontology. We have also implemented a translation tool to convert an XPDL model to its corresponding oXPDL representation, serving as a basis for further analysis

## 2 Related work

Our work is partially based on PSL [7], a first-order logic process ontology: the PSL axiomatisation can be used for behavioural reasoning in our ontology. In contrast to our work, PSL does not facilitate integration and reuse of other ontologies with background knowledge and Web ontologies, a consequence of the PSL knowledge representation formalism. As a result, process analysis with PSL cannot reuse definitions in external standardised business ontologies such as RosettaNet.

Other process interchange formats include the Petri Net Markup Language [2], the XML Metadata Interchange [20] specification and the Graph eXchange Language [28]. However, these have only informal semantics, focus on control-flow with little attention to data modelling or organisational modelling, and do not integrate with background knowledge; as such, they are unable to cover most of the competency questions listed previously, since these rely on knowledge outside of the direct control-flow model.

Business intelligence techniques based on process models have been proposed by Grigori *et al.* [6], who describe a process mining toolset on top of HP's Process Manager. The toolset supports quality managers through process analysis, prediction, monitoring, control, and optimisation features. However, the approach is limited to deadline estimation exception prediction, does not propose a model for more generic business intelligence operations and does not address the integration challenges that arise during analysis. Recent work by de Medeiros *et al.* [17] leverages classical process mining techniques to a semantic level by linking discovered results to external higher-level concepts. This work can be seen as complementary to ours: their focus rests on the mining techniques while we focus on representing and reusing process information and other business information.

## 3 Ontology engineering methodology

We have divided the ontology engineering process of oXPDL into four phases. First, we use a top-down engineering approach and represent all XPDL elements in our ontology language, explicitly formalising the informal semantics described in the XPDL standard specification. We then validated

the representation and internal consistency of the ontology by manually instantiating sample XPDL models in our ontology. Next, we defined transformation rules that lift a given XPDL document into its corresponding oXPDL representation, and implemented these transformations in our translation tool. Finally, we enriched the oXPDL ontology with relations that are implicit in the XPDL model and by relations required for answering our competency questions.

### 3.1 Ontology language

Several languages have been developed to model and represent knowledge in Web ontologies. These languages form the basis for the Semantic Web [1], an online interlinked Web of machine-readable information. The fundamental data-model of the Semantic Web is RDF [14], a language for asserting statements about arbitrary resources. The use of URIs allows statements from different sources to interlink, forming a hypergraph of statements. RDF Schema [3] is a vocabulary for RDF for defining classes, properties, and their hierarchies. OWL [16] and WSML [4] are expressive ontology languages, partially layered on RDF(S), to describe richer relations and information constraints.

Our general transformation and modelling approach is agnostic and independent of the choice for a particular ontology language. Without loss of generality for the overall approach, we choose the WSML ontology language as modelling language and output of the XPDL translator. This choice is based on the fact that WSML enables the integration of the process ontology with arbitrary (public) background ontologies and that a reasoning and an execution architecture is available to query and analyse the translated process descriptions. The WSML ontology language is actually a family of several language variants, which differ in their logical formalisms and levels of logical expressiveness; oXPDL uses the WMSL-Rule language variant. As mentioned before, the ontology and translation could also be implemented with another Web ontology language and semantic Web service framework.

### 3.2 Key modelling decisions

Before introducing the actual oXPDL ontology, we outline the general assumptions and the modelling approach taken.

**General mapping** Each XPDL modelling element (*XSD complex type*) is mapped to an ontology concept. Sub-elements with a *simple built-in type* are mapped to attributes with a similar built-in type in the ontology language, or if such built-in is not available, to a mapped simple type definition. A *complex type sub-element* is represented by creating a corresponding sub-concept and referencing that subconcept through an attribute in the parent concept, as shown in figure 2.

**Sequences and choices** *XSD choice* elements are translated into a constraining axiom which restricts instances to the specified set of types. *XSD sequences*, which specify that child elements must appear in a fixed order, are ignored since these are syntactical restrictions without conceptual meaning.

**Identifiers** For concept and attribute identifiers in the ontology, the XPDL element names are used directly. XPDL also uses *id* attributes of type *Token* to identify element types. These attributes are omitted in the ontology since all concepts and attributes in Web ontology languages are already identified through URIs. If no element type identifier is given in the XPDL instance, unique identifiers are created during translation based on the element name and timestamp.

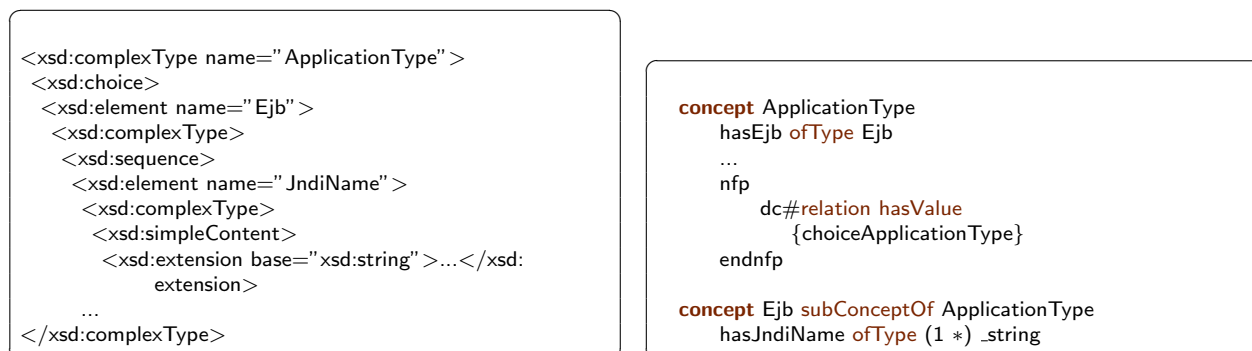


Figure 2: Complex extension type and its representation in oXDPL.

## 4 oXPDL ontology

Having discussed the general modelling approach we now present the actual oXPDL ontology and the approach for automatically deriving oXPDL ontology instances from XPDL models. The ontology covers all of XPDL and consists of around 125 concepts and some 50 logical axioms which extend the formalisation in our earlier m3po ontology [11] and PSL [7]. As explained, the ontology is not designed for manual process modelling: our tool automatically translates exported process instances to oXPDL and allows analysis through query answering and reasoning.

For readability we present the ontology in five sections, focusing on respectively the functional, control-flow, informational, organisational, and operational aspects [13]. We illustrate each section with sample ontology snippets based on our motivating example and show how each competency question can be answered. The complete ontology and the translation tool can be found online<sup>4</sup>.

### 4.1 Functional Aspect

The functional aspect provides modelling facilities to define workflow hierarchies and generic properties of workflow models. It can be seen as a meta-concept defining the reusability characteristics of the modelling elements in the underlying aspects.

In XPDL the *package* element type references all functional properties in the XPDL model. It comprises properties such as the *TypeDeclaration*, *DataField*, *PartnerLinkType* and *MessageFlow*. Although these properties give one the means to model generic properties of a process model, they are all related to the individual aspects described in the following paragraphs. The only generic functional properties in the XPDL model are the *PackageHeader* information, the hierarchical layering of workflows and the properties related to the graphical layout to represent a BPMN model or a proprietary graph-based model.

<sup>4</sup><http://www.m3pe.org/oXPDL/>

```

instance PackageType_1 memberOf PackageType
  hasName hasValue _string(" sample workflow process")
  hasPackageHeader hasValue PackageHeader_1
  hasTypeDeclaration hasValue Order
  hasParticipant hasValue DBConnection
  hasProcessType hasValue ProcessType1
  hasProcessType hasValue ProcessType1

instance PackageHeader_1 memberOf PackageHeader
  hasXPDLVersion hasValue _string("0.09")
  hasVendor hasValue _string("XYZ, Inc")
  hasCreated hasValue _string("6/18/2002 5:27:17 PM")

```

Listing 1: Sample *packageType* and *PackageHeader* instance

Listing 1 shows the translation of an XPDL example header into oXPDL. To answer our first competency question (all processes in which members of an organisation participate), we can run the query shown in Listing 2 on the translated process model.

```

?x[hasParticipant hasValue ?y] memberOf PackageType and ?y memberOf
Organisation

```

Listing 2: Answering competency question #1

Some properties introduced in XPDL 2.0 to represent BPMN diagrams (including graphical layout) have weak semantics. For example, the *pool* element type references process types and participants as strings. In oXPDL the range of these properties are *Participants* and *ProcessTypes*. The translation algorithm therefore performs a type check when creating *pool* instances by trying to match the string in the pool element with translated objects of type *Participant* or *ProcessType*. If no matches are found, the type of the pool element cannot be determined automatically, but could be added manually to the derived ontology.

Similar semantic problems are raised in the case of the *lane* element type which references a *ParentLane* and a *ParentPool* only by a string type. Again, a type check is implemented during translation to automatically derive object references if available. Figure 3 shows instances of pools and lanes in XPDL and oXPDL respectively. With the explicit referencing of *ProcessTypes* the model can be queried to answer the second competency question (which activities belong to Pool3) with the query shown in Listing 3.

```

<Pool Id="Pool3" Name="Pool1">
  <Process>ProcessType2</Process>
  <Lanes>
    <Lane Id="Lane3" Name="Lane-3">
      <ParentLane>Lane1</ParentLane>
    ...
  </Lanes>
</Pools>

```

```

hasProcess hasValue ProcessType2
hasName hasValue _string("Pool1")
hasLane hasValue Lane3

```

```

instance Lane3 memberOf Lane
hasName hasValue _string("Lane-3")
hasParentLane hasValue Lane1

```

Figure 3: Pools and lanes in XPDL and oXPDL

```

Pool3[hasProcess hasValue ?a] memberOf Pool and ?a[hasActivity hasValue ?b] memberOf ProcessType

```

Listing 3: Answering competency question #2

## 4.2 Control Aspect

The *Process* type, the primary type in XPDL, represents a process model and its constituent elements. It groups related activities, data, and resources together. *Activities* in XPDL represent the reusable task behaviour in a process and take one of the following types, a triggered event, a route activity that constrains the ordering of activities or a block activity. In oXPDL all activity types are modelled accordingly.

```

ProcessType
  hasName hasValue _string(" PIP3A1")
  hasActivity hasValue Activity1
  hasActivity hasValue Activity5
instance ProcessType2 memberOf ProcessType
  hasName hasValue _string(" PIP3A4")
  hasActivity hasValue Activity48
  hasActivity hasValue Activity49
  hasActivitySet hasValue ActivitySet2
instance ActivitySet1 memberOf ActivitySet
  hasActivity hasValue Activity48
  hasActivity hasValue Activity5
instance ActivitySet2 memberOf ActivitySet
  hasActivity hasValue Activity49

```

Listing 4: Subprocesses and their associated activities

```

ProcessType3[hasActivity hasValue ?a] memberOf ProcessType and
?b[hasActivity memberOf ?a] memberOf ActivitySet

```

Listing 5: Answering competency question #3

Listing 4 shows an example snippet from our translated motivating example process including the two workflows (PIP3A1 and PIP3A4) and subprocesses used within these processes. If we query for one of our competency questions (which activities belong to a subprocess), as shown in Listing 5, the answers contain not only activities belonging to *ActivitySets* which are directly linked from the *ProcessType3*, but also activities which belong to an *ActivitySet* in any other *ProcessType*. *ActivitySets*, equivalent to a BPMN embedded subprocess, can be used to group activities. This query is for example useful for a process modeler to know in what context the activity is used in other workflow models.

More expressive relations are added to oXPDL in support of the routing activities. These routing activities, one possibly type of activities, support routing decisions among the incoming transitions and/or among the outgoing transitions. Route Activities are also used to represent BPMN gateways. The semantics of Routing activities is given in section 4.6. The actual control flow in XPDL is modelled via explicit transitions between elements, including the Routing Activities modelling control flow constructs such as *Loop* and *Split*. This modelling paradigm closely resembles the behavioural definitions in BPMN, which as a graph based modelling notation relies on connectors between activities. Accordingly, the definition of control flow links in our ontology is modelled with a *transition* class, representing the arcs as a first class entity, with associated *hasFrom* and *hasTo* properties.

Activities may also represent events, equivalent to the BPMN event type. Events are independent of the control flow, and effect the flow of the process and usually have a cause (trigger) and/or an impact (result). There are three types of events, based on when they affect the flow: *StartEvents*, *IntermediateEvents*, and *EndEvents*, all modelled as subclasses of the Event class in oXDPL.

Since events are not directly modelled in the control flow, but can always trigger changes to the actual execution behaviour, one might be interested to query for all events that may affect the execution dependencies, as raised in our fourth competency question, supported in oXPDL with the query shown in Listing 6.

```
ProcessType1[hasActivity hasValue ?a] memberOf ProcessType and
?a[hasEvent hasValue ?b] memberOf Activity
```

Listing 6: Answering competency question #4

### 4.3 Informational Aspect

The informational aspect deals with data production and data consumption, i.e. the data flow between tasks. We provide a direct mapping from all data types to the ontology. The *DeclaredType*, *BasicType*, *EnumerationType*, *ExternalReference*, *ListType* and *ArrayType* are all subclasses of the *DataType* concept. The datatypes itself are however directly represented as built-in datatypes in the ontology language which inherit the type hierarchy from XSD schema.

However, more importantly, the modelling of messages can significantly benefit from an ontological modelling approach. Message flow is the predominant modelling paradigm in all web service flow description languages, like BPEL, but also an important modelling paradigm in semantic Web Service standards. XPDL offers a number of different possibilities to model Message Flow.

First, for collaborative processes, a Message Flow can be defined on the *PackageType* level. This modelling of messages between Pools defined at Package level is intended to encode BPMN semantics. Since, the attribute types *hasSource* and *hasTarget* in the *MessageFlow* concept in oXPDL are modelled as references to the *Pool* concept, we can include rules in the ontology to encode that a valid collaborative Message Flow always connects two separate Pools representing two participants in the process. Further, we can query the model for our fifth competency question (which messages are received by pool3) with the query shown in Listing 7. Through the reference of the Pool in the Package and its associated Participant, we also know who in particular has to implement this Message Exchange pattern.

```
?a[hasSource hasValue Pool3] memberOf MessageFlow and
?b[hasMessageFlow hasValue ?a] memberOf PackageType
```

Listing 7: Answering competency question #5

Whereas messages in XPDL are only referenced by an external identifier, in oXPDL such identifiers can point to an external message ontology. This allows one not only to use the type hierarchy of the message in the evaluation of expressions or in the oXPDL mapping class, but control flow or event triggers can actually be based on an evaluation of the message content. For our motivating example we can reuse a RosettaNet message ontology to reference the types of the PIP3A1 and PIP3A4 messages. This approach however, requires that there is an infrastructure in place that translates the actual XML messages used in the communication to ontology instances as described in [9]. If such an approach is followed and the RosettaNet ontology is imported, we can query our model for the sixth competency question (does a message in PIP3A4 include some substitutable product) with the query shown in Listing 8.

```
PIP3A4_send[hasMessage hasValue ?a] memberOf TaskSend and
?a[hasIdentifier hasValue ?b] memberOf MessageType and
?b[pip3a4:hasSubstituteProductReference hasValue pip3a4:ProductLineItem]
memberOf pip3a4:ProductLineItem
```

Listing 8: Answering competency question #6

To distinguish between concepts in oXPDL and the RosettaNet ontology we used a different namespace identifier (pip3a4) for the latter.<sup>5</sup> This query is useful to determine if the provider (the process owner in our motivating example) quotes a substitutable product that is a replaceable to the initially requested product.

#### 4.4 Organisational Aspect

The organisational aspect defines *who* is responsible to perform a task in a workflow and gives the modeler means to assign constraints on the agent (be it a human or a machine) responsible for carrying out a task.

The organisational aspect in an XPDL document is only weakly defined; a process participant is simply a token of the following types: resource set, resource, organisational unit, role, human, or system. In oXPDL these types are modelled as independent classes based on representations in the Suggested Upper Merged Ontology (SUMO) [19], a richly axiomatised formal ontology.

For the additional relations in the organisational model of oXPDL we follow a reference model defined in m3po [11], which is based on the model introduced in [18]. We define the *Participant* to be either an organisational or human process resource. The *belongsTo* attribute allows to hierarchically structure *OrganisationalUnits* and *Persons*. *Competence* describes the possible actions a *Participant* is permitted to perform. A *Qualification* is another direct property of a *Person*. The Person Class, among other things, also references a FOAF<sup>6</sup> profile with the *seeAlso* property. Modelling such information allows us to query the knowledge base for our seventh competency question (in which processes does Armin Haller participate) with the query shown in Listing 9.

```
?a[hasParticipant hasValue ?b] memberOf ProcessType and
?b[hasParticipantType hasValue ?c] memberOf Participant and
?c[hasType hasValue ?d] memberOf ParticipantType and ?d[seeAlso
hasValue ."http://sw.deri.org/~haller/foaf.rdf"] memberOf Person
```

Listing 9: Answering competency question #7

#### 4.5 Operational Aspect

The operational aspect deals with the modelling of applications to be managed and invoked by the WfMS. These applications can be very different in nature, but the technical details of the application programs are always kept transparent in the workflow model. Although XPDL models different application types, such as EJBs, Java Objects, XSLT scripts, Forms and Business Rules, the most important modelling element are Web Services. The semantics of all former application types are kept weak in the translation to oXPDL and only their URI references were changed from string types to IRIs in the ontology language. However, the Web Service modelling was enriched as follows.

<sup>5</sup>We simplified the query for illustrative purposes; the actual concept hierarchy in the RosettaNet ontology is deeper and the conjunctive query would have to span multiple classes.

<sup>6</sup><http://xmlns.com/foaf/0.1/>

First, the associated Input and Output messages are referenced as a type *Message*. Second, *PartnerLinks* and *PartnerLinkTypes* are explicitly linked, similarly to the modelling of Web Services in BPEL. In oXPDL the *PartnerLink* modelled in the *PackageHeader*, as described in section 4.1, references this *partnerLinkType* and defines which role is taken by the process itself and which role is taken by a partner. In this way, the *partnerLinkType* describes a contract between two partners in terms of their roles (which are in oXDPL also related to the Pool concept) and the corresponding WSDL *portTypes* the partners have to provide. An example of the modelling of Web Services in XPDL and its representation in our ontology is given in listing 10.

```

Application_Repository_WS5 memberOf Application
  hasName hasValue _string(" receive PIP3A4")
  hasType hasValue webservice_PIP3A4
instance webservice_PIP3A4 memberOf ApplicationType
  hasWebService hasValue webservice_PIP3A4_receive
instance webservice_PIP3A4_receive memberOf WebService
  hasWebServiceOperation hasValue PurchaseOrderRequestAction_1
  hasInputMsgName hasValue PIP3A4_request
instance PurchaseOrderRequestAction_1 memberOf WebServiceOperation
  hasPartner hasValue Partner_1
  hasService hasValue Service_1
instance Partner_1 memberOf Partner
  hasPartnerLinkType hasValue PIP3A4_seller
instance PIP3A4_seller memberOf PartnerLinkType
  hasRoleType hasValue PIP3A4_seller_role
instance Service_1 memberOf Service
  hasPortName hasValue _string(" PurchaseOrderRequestActionPortType")

```

Listing 10: Application Type Web Service in oXPDL

Through the explicit referencing of messages and the *PartnerLinkType* as shown in the listing we can query the model for the eight competency question (which partners should implement a web service operation in ProcessType2) with the query shown in Listing 11.

```

ProcessType2[hasApplication hasValue ?a] memberOf ProcessType and
?a[hasType hasValue ?b] memberOf Application and
?b[hasWebService hasValue ?c] memberOf ApplicationType and
?c[hasWebServiceOperation hasValue ?d] memberOf WebService and
?d[hasPartner hasValue ?e] memberOf WebServiceOperation

```

Listing 11: Answering competency question #8

## 4.6 Behavioural Axioms

In XDPL as well as in the proposed ontologisation in oXPDL, the *route activities* together with transitions between them constrain the ordering of activities in a process model. However, the semantics of XPDL routing constructs are only defined textually in the specification document. Thus, a direct translation of an XPDL document can only represent the structural elements of a process model. To reason upon behavioural properties in the ontology, axioms defining the behavioural semantics have to be added to the model. One can adopt the axiom schemata from PSL [7] to reason upon behavioural properties in oXDPL. By relating the ordering constraints in oXPDL to the relations constraining the ordering of activities in PSL, we can query the model for behavioural properties of the model. For space consideration we refer to [10], where we have shown how to relate similar routing constructs as used in oXPDL to the occurrence and activity trees in PSL. In this paper, however, since our model is intended to be used as a way to publish process

$$\begin{aligned}
T1 ::= T2 &:- \text{before}(T1, T2) \wedge \text{before}(T2, T1) \\
\text{before}(T1, T3) &:- \text{before}(T1, T2) \wedge \text{before}(T2, T3) \wedge \\
&\quad \neg(T1 ::= T2) \wedge \neg(T1 ::= T3) \wedge \neg(T2 ::= T3)
\end{aligned}$$

Table 1: Axioms constraining *before* relation

$$\begin{aligned}
\text{earlier}(S1, S3) &:- \text{earlier}(S1, S2) \wedge \text{earlier}(S2, S3) \\
\text{earlier}(S1, S3) &:- \text{earlier}(S1, S2) \wedge \text{earlier}(S3, S2) \wedge \\
&\quad \neg\text{earlier}(S3, S1) \wedge \neg(S3 ::= S1) \\
S3 ::= S1 &:- \text{earlier}(S1, S2) \wedge \text{earlier}(S3, S2) \wedge \\
&\quad \neg\text{earlier}(S1, S3) \wedge \neg\text{earlier}(S3, S1)
\end{aligned}$$

Table 2: Axioms constraining *earlier* relation

models on the Semantic Web and to analyse process instances exchanged between collaboration partners, we are primarily interested in the ordering relations of running or finished processes. The ordering in a so called workflow log is always totally ordered, since during execution only one possible path has been taken. For this axiomatisation we have adopted the occurrence tree theory extension of PSL. Our model includes mainly the axioms of PSL defining a discrete ordering of timepoints. A transitive, irreflexive and totally ordered relation, *before*, is introduced for that purpose. The listing below shows the axioms related to the *before* relation.

An occurrence tree is a set of all discrete sequences of activity occurrences. The tree is constrained by the *earlier* and *initial* relation.

A *concrete* relation is introduced to constrain the tree to that occurrence ordering which actually occurred during execution. When an XPDL process is converted to oXPDL, concrete relations are added to the ontological model according to the execution order of the activities in the log based on their time occurrence. The set of concrete *activityOccurrences* constitute a single branch in the occurrence tree.

To be able to do that we need to model process instances according to an XPDL model in the ontology as well. However, since all process aware information systems log events in different formats, it is necessary to translate their process instances to oXPDL. In [8] we present an approach how to automatically translate various workflow logs to oXDPL.

Based on that model we can query the model for the competency question #9 (Which tasks precede the ‘‘Prepare Quote’’ in the instances of the ‘‘PIP3A1’’ process) as follows.

```
?a[isOccurrenceOf hasValue ProcessType1] memberOf ProcessOccurrence
and ?a[hasActivityOccurrence hasValue ?b] memberOf ProcessOccurrence
and ?a[hasActivityOccurrence hasValue ?c] memberOf ProcessOccurrence
and ?b[isOccurrenceOf hasValue PrepareQuote] memberOf
ActivityOccurrence and relationInstance before(?c,?b) and
relationInstance concrete(?c,?b).
```

Listing 12: Answering competency question #9

By inference on the axiom schemata, the reasoner interpreting the query returns all *ActivityOccurrences* preceding *PrepareQuote*, also those which are not immediate predecessors and directly linked via a transition relation. Similarly we can query the model for competency question #10 (What is the most frequent path in the “PIP3A1” processes?). It returns all possible occurrence paths. The statistical analysis would have to be done external to the reasoner.

```
?a[isOccurrenceOf hasValue ProcessType1] memberOf ProcessOccurrence
and ?a[hasActivityOccurrence hasValue ?b] memberOf ProcessOccurrence
and relationInstance before(?b,?b) and relationInstance
concrete(?b,?b).
```

Listing 13: Answering competency question #10

## 5 XPDL2WSML converter implementation

The process model conversion involves two steps. Firstly, the oXPDL Core Ontology (CO) is loaded, serving as the language grammar including the concept hierarchy, its attributes and their cardinalities and secondly, the XPDL instance are analysed. The XML file is parsed to find children and sibling nodes. Every node is analysed, whether it is a *text node* or an *element node*. In both cases it is further checked if the concept or attribute is defined in CO. If it is, an attribute for the current instance or a new instance is created. The names of *element nodes* are mapped with a special dictionary including synonyms defined by the WfMC [24], to be able to map element names to the concepts in CO. After the name mapping the converter checks whether a concept is actually defined in CO or not. If yes, the process continues with the instance name construction. There are two cases that may occur. Either the ID is explicitly defined in the XML document or the tool generates a unique identifier by a combination of the element name and a hash of the creation date. When a new instance is created all attributes within the concept are parsed, their types are checked and if required they are added to the ontology.

The result of the conversion process is a knowledge base populated with instances that represent the knowledge modelled in XPDL, described in terms of our oXPDL ontology. The complete conversion algorithm of an XML instance according to the XPDL schema is illustrated in the activity diagram in figure 4.

We have verified the correctness of the translation tool with respect to the defined oXPDL Core Ontology. The translation procedure is not hard-coded into the converter tool, but instead based on the grammar described in the Core Ontology and a set of transformation rules, making the converter flexible with respect to modelling changes or extensions to the oXPDL ontology: if changes to the Core Ontology are made, rules can be added to encompass those changes.

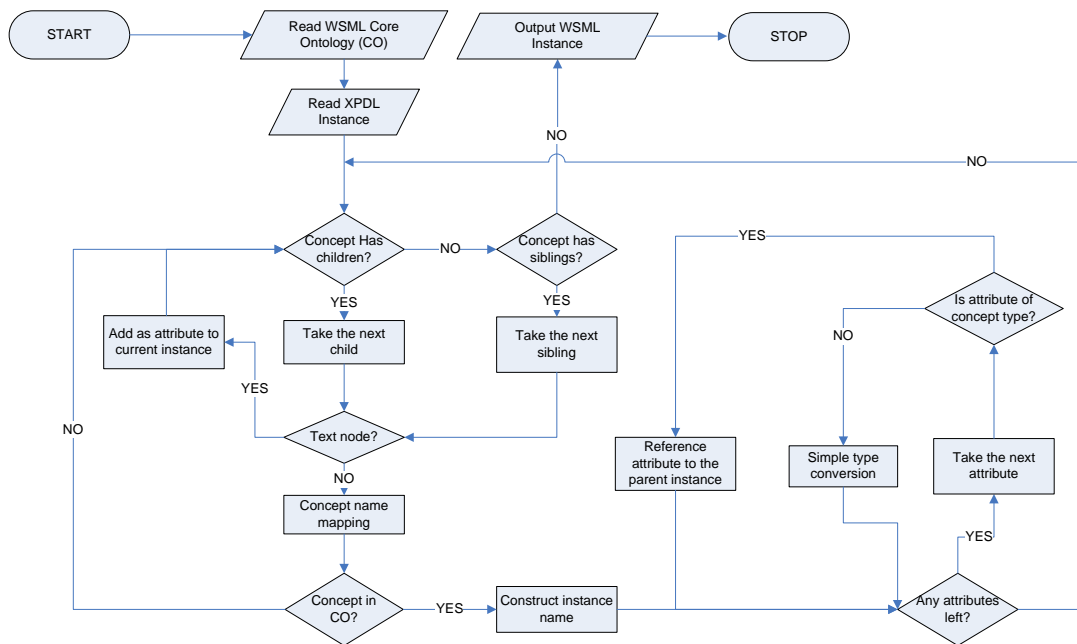


Figure 4: Processing steps of the conversion algorithm

## 6 Conclusion

Inter-organisational workflows between collaborating business partners as seen in virtual enterprises or multi-partner Web service choreographies, require the exchange and interpretation of various process models. As we have shown, XPDML is a good starting point for process interoperability, but lacks explicit and formal semantics, and the ability to interlink to existing knowledge bases with background knowledge.

We have presented our approach to ontologise the complete XPDML standard, resulting in our oXPDML ontology, and a translation tool that transforms XPDML models into valid oXPDML instance documents. We have explained the ontologisation methodology, based on a guiding set of “competence questions”. The resulting oXPDML process models may be used for integrated process analysis, by querying and reasoning over multiple models, each of which may originate from different information systems, in combination with business rules described in background ontologies.

## References

- [1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
- [2] J. Billington, S. Christensen, K. M. van Hee, E. Kindler, *et al.* The petri net markup language: Concepts, technology, and tools. In *24th International Conference on Applications and Theory of Petri Nets (ICATPN)*, vol. 2679 of *Lecture Notes in Computer Science*, pp. 483–505. Springer, Eindhoven, The Netherlands, 2003.

- [3] D. Brickley and R. Guha, (eds.) *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation, 2004.
- [4] J. de Bruijn, H. Lausen, A. Polleres, and D. Fensel. The web service modeling language WSM: An overview. In *Proceedings of the European Semantic Web Conference (ESWC)*, pp. 590–604. 2006.
- [5] M. Dumas, W. M. P. van der Aalst, and A. H. ter Hofstede. *Process Aware Information Systems: Bridging People and Software Through Process Technology*. John Wiley & Sons, Ltd, Hoboken, New Jersey, 2005.
- [6] D. Grigori, F. Casati, M. Castellanos, U. Dayal, *et al.* Business process intelligence. *Computers in Industry*, 53(3):321–343, 2004.
- [7] M. Gruninger. Ontology of the process specification language. In S. Staab and R. Studer, (eds.) *Handbook on Ontologies*, pp. 575–592. Springer-Verlag, 2004.
- [8] A. Haller, W. Gaaloul, and M. Marmolowski. Analysis and mining of ontological process model instances. Tech. Rep. DERI-TR-2007-03-28, DERI, 2008.
- [9] A. Haller, J. Gontarczyk, and P. Kotinurmi. Towards a complete SCM Ontology – The Case of ontologising RosettaNet. In *Proceedings of the 23th Annual ACM Symposium on Applied Computing (SAC2008)*, pp. 1467–1473. ACM Press, 2008.
- [10] A. Haller and E. Oren. m3pl: A Work-FLOWS ontology extension to extract choreography interfaces. In *Proceedings of the Workshop on Semantics for Business Process Management at European Semantic Web Conference (ESWC2006)*. Budva, Montenegro, 2006.
- [11] A. Haller, E. Oren, and P. Kotinurmi. m3po: An Ontology to Relate Choreographies to Workflow Models. In *Proceedings of the 3rd International Conference on Services Computing*, pp. 19–27. IEEE Computer Society, Chicago, Illinois, USA, 2006.
- [12] M. Hepp. Products and services ontologies: A methodology for deriving owl ontologies from industrial categorization standards. *International Journal on Semantic Web & Information Systems*, 2(1):72–99, 2006.
- [13] S. Jablonski. Mobile: A modular workflow model and architecture. In *Proceedings of the International Working Conference on Dynamic Modelling and Information Systems*. Nordwijkerhout, The Netherlands, 1994.
- [14] G. Klyne and J. J. Carroll, (eds.) *Resource Description Framework: Concepts and Abstract Syntax*. W3C Recommendation, 2004.
- [15] D. Martin, M. Burstein, J. Hobbs, O. Lassila, *et al.* Owl-s: Semantic markup for web services. Member submission, W3C, 2004. Available from: <http://www.w3.org/Submission/OWL-S/>.
- [16] D. L. McGuinness and F. van Harmelen, (eds.) *OWL Web Ontology Language*. W3C Recommendation, 2004.
- [17] A. K. A. de Medeiros, C. Pedrinaci, W. M. P. van der Aalst, J. Domingue, *et al.* An outlook on semantic business process mining and monitoring. In *OTM Workshops*, pp. 1244–1255. Vilamoura, Portugal, 2007.
- [18] M. zur Muehlen. Organizational management in workflow applications issues and perspectives. *Information Technology and Management*, 5(3-4):271–291, 2004.

- [19] I. Niles and A. Pease. Towards a standard upper ontology. In *Proceedings of the International Conference on Formal Ontology in Information Systems*, pp. 2–9. New York, NY, USA, 2001.
- [20] OMG. Mof 2.0/xmi mapping specification, v2.1. 2001.
- [21] D. Roman, U. Keller, H. Lausen, J. de Bruijn, *et al.* Web Service Modeling Ontology. *Applied Ontologies*, 1(1):77 – 106, 2005.
- [22] 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.
- [23] S. Thatte *et al.* Business process execution language for web services, v1.1. 2003.
- [24] The Workflow Management Coalition. Terminology and glossary. 1999. Available from: <http://www.wfmc.org/standards/docs/tc003v11.pdf>.
- [25] The Workflow Management Coalition. Workflow Standard Process Definition Interface – XML Process Definition Language. Tech. Rep. WFMC-TC-1025, WfMC, 2005. Available from: [http://www.wfmc.org/standards/docs/TC-1025\\_xpd1\\_2\\_2005-10-03.pdf](http://www.wfmc.org/standards/docs/TC-1025_xpd1_2_2005-10-03.pdf).
- [26] M. Uschold and M. Grüninger. Ontologies: principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.
- [27] S. A. White *et al.* Workflow Management Coalition. Workflow Standard Process Definition Interface – XML Process Definition Language. OMG Final Adopted Specification, OMG, 2006.
- [28] A. Winter and C. Simon. Using GXL for exchanging business process models. *Inf. Syst. E-Business Management*, 4(3):285–307, 2006.