XML Representation of Process Descriptions

Joshua Lubell
National Institute of Standards and Technology (NIST)
Manufacturing Systems Integration Division [http://www.nist.gov/msid]

<lubell@nist.gov>

Last revised $Date: 2002/04/18 20:58:51 $

I consider the use of XML (Extensible Markup Language) for applications involving discrete processes. Examples of such processes include production scheduling, process planning, workflow, business process re-engineering, simulation, process realization process modeling, and project management. I begin by considering what process meta data is and how it can be used, before looking at an example of how to design an XML vocabulary for the exchange of process meta data.

Notes

- This paper's content was originally published in Chapter 14 (“Process Descriptions”) of *Professional XML Meta Data*, Wrox Press, 2001, ISBN 1861004516.
- Commercial equipment and materials are identified in order to describe certain procedures. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose. Unified Modeling Language, UML, Object Management Group, and other marks are trademarks or registered trademarks of Object Management Group, Inc. in the U.S. and other countries.
- Links to non-NIST websites are provided because they have information that may be of interest to readers of this paper. NIST does not necessarily endorse the views expressed or the facts presented on these sites. Further, NIST does not endorse any commercial products that may be advertised or available on these sites.

What is Process Data?

This article discusses the use of XML (Extensible Markup Language) for applications involving *discrete processes*, i.e. processes described as individually distinct sequences of events. Examples of such processes include production scheduling, process planning, workflow, business process re-engineering, simulation, process realization, process modeling, and project management.

I do not address *process characterization*, which I define as the representation of a process independent of any specific application. For example, I am not concerned with representing a process's dynamic or kinematic properties, such as tool chatter, or numerical models capturing limits on a process's performance. Since process characterizations employ methods (e.g. differential equations) that are quite different from the techniques used to describe sequences of discrete events, a discussion on representing process characterizations in XML would require its own article (if not its own book).

Many applications use process descriptions. The problem is that these applications work with process descriptions in their own internal representations. Therefore communication between them, a growing need for industry, is nearly impossible without some kind of translator. Since an environment with n applications needing to exchange data potentially requires n*(n-1) translators, we clearly need some sort of exchange standard for process data in order to avoid having to implement a point-to-point translator for every pair of applications.

XML is a natural syntax choice for the exchange of process data. XML defines a well-tested and extensible standard syntax for representing structured data. XML is text-based and non-proprietary. Processing software for XML is widely available. However, any useful XML formulation of process information should be based upon well-defined semantics. One such definition, which I will use in this chapter as a source of semantics for process representation, is the Process Specification Language (PSL). When I discuss how to use XML to represent process data later on in this chapter, PSL semantics will influence my guidelines, although (to keep matters simple) I won't attempt to formally derive my XML vocabulary from PSL.

The PSL project [http://www.nist.gov/psl] began as a collaborative effort led by the National Institute of Standards and Technology (NIST) [http://www.nist.gov], a US government agency working with industry to develop and apply technology, measurements, and standards. The project's goal is to create a standard language for process specification, in order to integrate multiple process-related applications throughout their life cycle. PSL has recently been proposed as an international standard (ISO 18629).

Although I choose PSL to guide this foray into the XML representation of process descriptions, PSL is not the only source of process semantics. Two other possible sources for semantics are the Unified Modeling Language (UML™) and the Workflow Management Coalition's Workflow Reference Model. I briefly discuss these later in the section called “Alternative Process Representation Approaches” near the end of this article.

About PSL

Most computer-interpretable languages are rigorous when it comes to specifying their syntax but fuzzy when it comes to semantics. Language specifications often describe syntax unambiguously using grammar rules, but state the meaning of the terms in the syntax using ambiguous prose. Unlike most languages, PSL has formally defined semantics. However, PSL doesn't specify a single syntax. Rather, PSL allows for the possibility of multiple syntaxes, with the choice of syntax depending on factors such as the nature of the process being described and the data source and destination.

Key to PSL are the formal definitions (ontology) that underlie the language. Because of these explicit and unambiguous definitions, information exchange can be achieved without relying on hidden assumptions or subjective mappings. PSL semantics are represented using Knowledge Interchange Format (KIF). Briefly stated, KIF [http://logic.stanford.edu/kif] is a formal language developed for the exchange of knowledge among disparate computer programs. KIF is a declarative language. It can express arbitrary logical assertions and rules, and it allows for the representation of knowledge about knowledge (meta knowledge). Thus KIF provides the level of rigor necessary to unambiguously define concepts, a necessary characteristic to exchange process information using the PSL ontology.

The primary components of PSL are KIF expressions constituting the PSL ontology for processes. The PSL ontology include an infinite set of terms, but they can only be shared if everyone agrees on their definitions. It is the definitions that are being shared, not simply the terms. For example, consider the KIF term `(between a b c)`. This term tell us that one thing is “between” two other things, whatever that means. To make the term meaningful, I need to supplement it with definitions, such as the following:

```
(defrelation between (?p ?q ?r) :=
   (and (before ?p ?q) (before ?q ?r)))
```

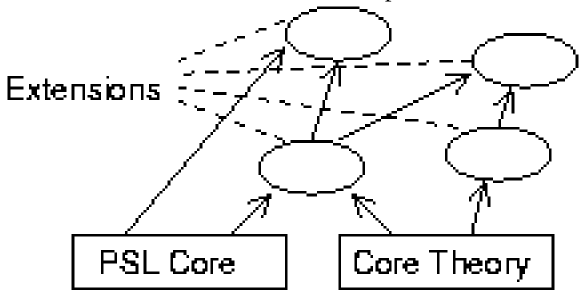
This KIF definition says that if you have three things: `?p`, `?q`, and `?r`, and that `?p` is before `?q` and `?q` is before `?r`, then `?q` is between `?p` and `?r`. The definition adds some semantics to “between”. Of course, it does not specify any semantics for “before”. The meaning of “before” could be specified in other definitions, or as is the case with PSL, “before” might be part of our primitive lexicon. The primitive lexicon consists of those terms for which I do not give definitions; rather, I specify axioms that constrain the interpretation of the terms. An example of an axiom constraining the interpretation of “before” is the statement that the “before” relation is irreflexive (i.e., nothing can be before itself). This axiom is formally specified in KIF as follows:

```
(forall (?p)
        (not (before ?p ?p)))
```

The PSL ontology [http://www.nist.gov/psl/psl-ontology] consists of primitive terms, definitions, and axioms for the concepts of PSL. However, this is not simply an amorphous set of sentences. The figure below gives an overview of the semantic architecture of the PSL ontology. There are three major components:

PSL Core	The most basic elements of the PSL ontology.
Core Theories	Very widely applicable extensions to PSL Core.
Extensions	Definitions capturing the semantics of process terminology for different applications.

The PSL ontology is organized modularly in order to facilitate the addition of new extensions as future industrial requirements for PSL emerge. PSL's modularity also makes it possible for applications to support a subset of extensions responding to a particular class of process specifications, without having to support the entire PSL ontology. The PSL semantic architecture is represented in the following diagram:



PSL Core specifies the concepts in the PSL ontology corresponding to the fundamental intuitions about activities. The extensions are organized by logical dependencies - one extension depends on another if the definitions of any terms in the first extension require terms defined in the second extension. PSL Core is therefore intended as the basis for defining terms of the extensions in the PSL ontology. PSL extensions often define new terms by specifying constraints on core terms.

Supplementing PSL Core is a set of core theories (also known as the “outer core”). This special set of extensions provides building blocks common to a wide variety of applications of PSL. Some of these extensions introduce new primitive concepts. They do so because the concepts introduced in PSL Core are not sufficient for defining the terms introduced in the extension. Therefore, new primitive concepts are introduced within the extensions to ensure that all other terms within the extension can be completely defined.

As I mentioned earlier, PSL does not mandate a single syntax for exchanging process descriptions. However, the PSL project proposes to standardize an XML framework for the exchange of process specifications. The framework has not yet been developed, but one idea suggested is to determine an appropriate XML vocabulary for an application based on meta data associated with the application describing the kinds of processes supported. The approach I take in the remaining sections of this chapter is less ambitious, but probably a good first step toward understanding how one might describe processes using XML. Rather than try to create a custom-built XML exchange syntax, I instead develop an (admittedly limited) XML vocabulary that captures some of the more fundamental concepts from the PSL ontology.

Basic Characteristics of Process Data

To guide our design of an XML vocabulary for process data, we'll begin by enumerating some key concepts defined in PSL core. These are:

Activity	A class or type of action. For example, painting a house is an activity. It is the class of actions in which houses are being painted.
Activity Occurrence	An event or action that takes place at a specific place and

	time, in other words an instance or occurrence of an activity. For example, painting John's house in Baltimore, Maryland at 2 PM on May 25, 2001 is an occurrence of the “paint house” activity.
Time Point	An instant separating two states, for example the point at which the first coat of paint is dry but before the second coat has been started.
Object	Anything that is not a time point or an activity, for example John's house.

Some other ideas, introduced in some of the more fundamental PSL extensions, are:

Ordering	Activities occur in ordered sequences delimited by time points.
Parallelism	Activities can occur at the same time.
Decomposition	Activities can contain sub-activities. Occurrences of such activities contain sub-occurrences.
Objects Versus Resources	A resource is an object being used by an activity. For instance, a paintbrush could be a resource of the “paint house” activity.

These are just some of the most basic PSL ontological concepts. There are many others including resource contention, states and conditions, complex ordering relationships, and many more.

An area not included in the PSL ontology, but nevertheless necessary for representing processes, is *object structure*. Since process specifications need to be able to describe an activity's resources, or at least refer to object descriptions specified elsewhere, PSL simply assumes that objects are described using methods outside of PSL's scope. In the guidelines for XML representation of processes discussed in the next section, we suggest using the Resource Description Framework (RDF) [http://www.w3.org/RDF] to describe objects.

XML Representation of Process Data

Now that I have extracted some process representation concepts from the PSL ontology, the next step is to come up with a set of guidelines for describing processes in XML. The business case for doing this is compelling. Vendors of mainstream software applications such as Internet browsers, database environments, and business productivity tools, are either already supporting or intend to support XML in their products. Mapping PSL instances to XML will enable process specifications to be interpreted by these generic applications, lowering the barriers to data sharing.

What XML Can and Cannot Do

An XML markup scheme for process data should take advantage of what XML does best, while minimizing the impact of where XML falls short. XML's “tag-centric” syntax makes it a natural fit for representing ordered sequences and hierarchies. Thus it is well suited for ordering time points and occurrences of activities. It is also good at representing sub-activities and sub-occurrences. Another capability of XML, useful for process representation, is XML's modularity. For example, using XML namespaces I can embed an arbitrary object description into a process specification and leave it up to a software tool, separate from the process specification interpreter, to parse the object description. I can also employ namespaces to modularize our process markup language itself (perhaps mirroring PSL's modularization).

Although XML has many advantages for representing processes, it has a major disadvantage. While XML excels as a serialization syntax for exchanging data structures between applications, XML is not very good at expressing the kinds of complex constraints needed for process descriptions. For example, it might be difficult for an XML schema for a process description language to enforce scheduling constraints involving shared. Such constraints could be more easily expressed in a rich language for knowledge representation such as KIF.

Because XML is deficient when it comes to representing complex constraints on populations of data elements, its process representation capabilities are limited. However, this does not mean that I cannot use XML to exchange process descriptions. Rather, it means that I probably would not want to exchange *all* of a process description's underlying ontology in XML, and I cannot count on an XML language to enforce *all* constraints on process data. It also means that XML would be a poor authoring environment for all but the most simple process descriptions.

Guidelines for Representing Process Data in XML

To maximize XML's strengths while minimizing its weaknesses, we suggest the following guidelines for representing process descriptions in XML. I do not claim that these guidelines are exhaustive or even optimal for the subset of process descriptions they cover. However, they may prove useful if you are new at developing XML process description vocabularies.

Use RDF to Represent a Process's Resources

RDF is probably the best choice for representing the objects used in a process. RDF can be embedded into process descriptions as needed, and RDF Schema is useful for specifying object structure, classes and instances, and inheritance relationships. An alternative to using RDF would be to write an XML "mini-language" to describe your objects, but it might be hard to develop a single mini-language that would satisfy all possibilities for objects in a process description.

As a matter of fact, RDF/RDF Schema could be used to represent not just the objects, but also the entire process description. However, these guidelines choose to limit the use of RDF to representing objects used in a process, mainly in order to make the XML syntax more human-readable.

Represent Time Points as Sequences of Elements

Represent time points as sequentially ordered groups of elements, with each time point element having a unique identifier. If the XML application uses a document type definition (DTD) or schema, the unique identifier should be represented using an ID attribute so that references to the time point can be made using IDREF, or alternatively using a Uniform Resource Identifier (URI) reference. Each time point element may optionally contain character data (intended for human consumption, not machine processing) documenting the meaning of the time point.

If I were to use these sample markup declarations:

```
<!ELEMENT  timepoints   (timepoint+)                                >
<!ELEMENT  timepoint    (#PCDATA)                                   >
<!ATTLIST  timepoint
             id         ID                                #REQUIRED >
```

An example of time point data might be:

```
<timepoints>
   <timepoint id="t1">start</timepoint>
   <timepoint id="t2">end</timepoint>
</timepoints>
```

Incidentally, I use DTD syntax in this chapter rather than World Wide Web Consortium (W3C) XML Schema [http://www.w3.org/XML/Schema] syntax because DTD syntax is less verbose and is adequate enough for our examples. XML Schema syntax may be a better choice for real world applications. See the section called "Choosing an XML Schema Language" for further discussion of this issue.

Create Hierarchies for Activities

For each activity, specify a unique identifier (with an ID attribute if using a DTD) and an activity name. If the activity contains sub-activities, specify these within a container element. If the activity has no sub-activities, specify the resources used with references to the appropriate class defined in the RDF Schema.

So our sample markup declarations might look like this:

```
<!ELEMENT  activity     (name, (activity+ | resource*))             >
<!ATTLIST  activity
             id         ID                                 #REQUIRED >
<!ELEMENT  name         (#PCDATA)                                    >
<!ELEMENT  resource     EMPTY                                        >
<!ATTLIST  resource
             rdf:resource
                        CDATA                              #REQUIRED >
```

Consider the activity of preparing a meal of leftovers. I represent this activity as having two sub-activities: reheating the leftover food and reading a newspaper (while waiting for the food to be ready). Let us further suppose that reheating requires two resources: an oven and a frozen meal, and that reading the newspaper requires a newspaper as a resource. Data representing this activity might appear as follows. The values of the `rdf:resource` attributes assume that `Oven`, `FrozenMeal`, and `Newspaper` are defined as classes in an RDF schema, the RDF schema and the activity data are in a file whose relative URI is `leftovers.rdf`, and that the entire process description is enclosed in a wrapper element as follows:

```
<processdesc xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

   ...

</processdesc>
```

Here is XML markup for the “prepare leftovers” activity:

```
<activity id="a4">
   <name>prepare leftovers</name>
   <activity id="a5">
      <name>reheat food</name>
      <resource rdf:resource="leftovers.rdf#Oven"/>
      <resource rdf:resource="leftovers.rdf#FrozenMeal"/>
   </activity>
   <activity id="a6">
      <name>read newspaper</name>
      <resource rdf:resource="leftovers.rdf#Newspaper"/>
   </activity>
</activity>
```

Create Hierarchies for Occurrences, and Allow for Parallelism

Specify occurrences of activities in sequential order with sub-activities enclosed inside parent activities. Each activity occurrence should have a beginning and ending time point unless they can be inferred from a parent activity occurrence. If the activity occurrence cannot be decomposed into sub-activities, it should contain a list of any RDF-defined resource instances it uses. References to time points and activities should refer to their respective unique identifiers (and should be of type IDREF if using a DTD).

A `fork` element, whose content is one or more activity occurrences, can represent activities taking place in parallel that start and end at the same time.

The markup declarations for this example could be:

```
<!ELEMENT  occurrence   ((occurrence | fork)+ | resource*)           >
<!ATTLIST  occurrence
           activity     IDREF                               #REQUIRED
           begin        IDREF                               #IMPLIED
           end          IDREF                               #IMPLIED  >
<!ELEMENT  fork         (occurrence+)                                 >
<!ATTLIST  fork
           begin        IDREF                               #IMPLIED
           end          IDREF                               #IMPLIED  >
```

An occurrence of the activity “prepare leftovers”, with reheating and reading taking place in parallel, might appear as follows. I assume that `myMicrowave` is an instance of `Oven`, `tunaCasserole` is an instance of `Frozen-Meal`, and that `2001May29WashingtonPost` is an instance of `Newspaper`.

```
<occurrence activity="a4" begin="t1" end="t2">
   <fork>
      <occurrence activity="a5">
         <resource rdf:resource="leftovers.rdf#myMicrowave"/>
         <resource rdf:resource="leftovers.rdf#tunaCasserole"/>
      </occurrence>
      <occurrence activity="a6">
         <resource rdf:resource="leftovers.rdf#2001May29WashingtonPost"/>
      </occurrence>
   </fork>
</occurrence>
```

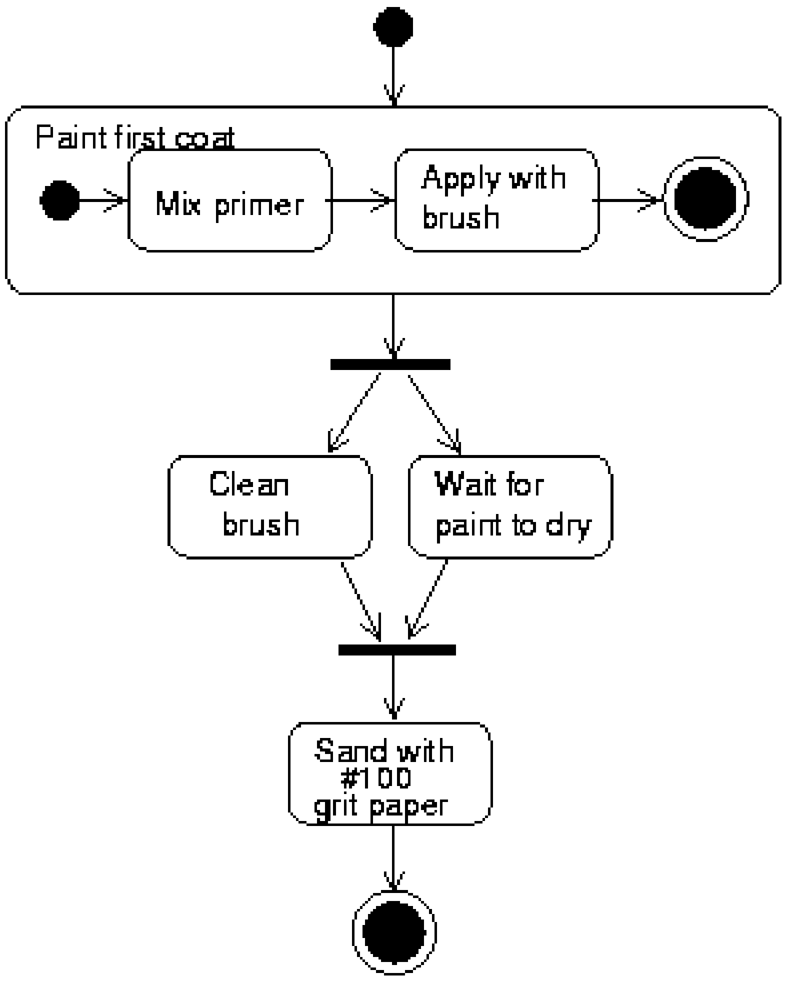
Omit Underlying Ontologies and Foundational Theories

As I discussed earlier in the section called “What XML Can and Cannot Do”, XML is not designed to be a general-purpose knowledge representation language. An XML representation of a process description should not attempt to encode the entire PSL ontology. For example, the axiom that “before” is irreflexive probably does not need to be part of an XML vocabulary for processes. Such axioms and other underlying definitions and assumptions should probably be omitted from the XML representation, unless they describe “containing” or “ordering” relationships that can be easily represented by XML markup.

Extended Example: A Finishing Process

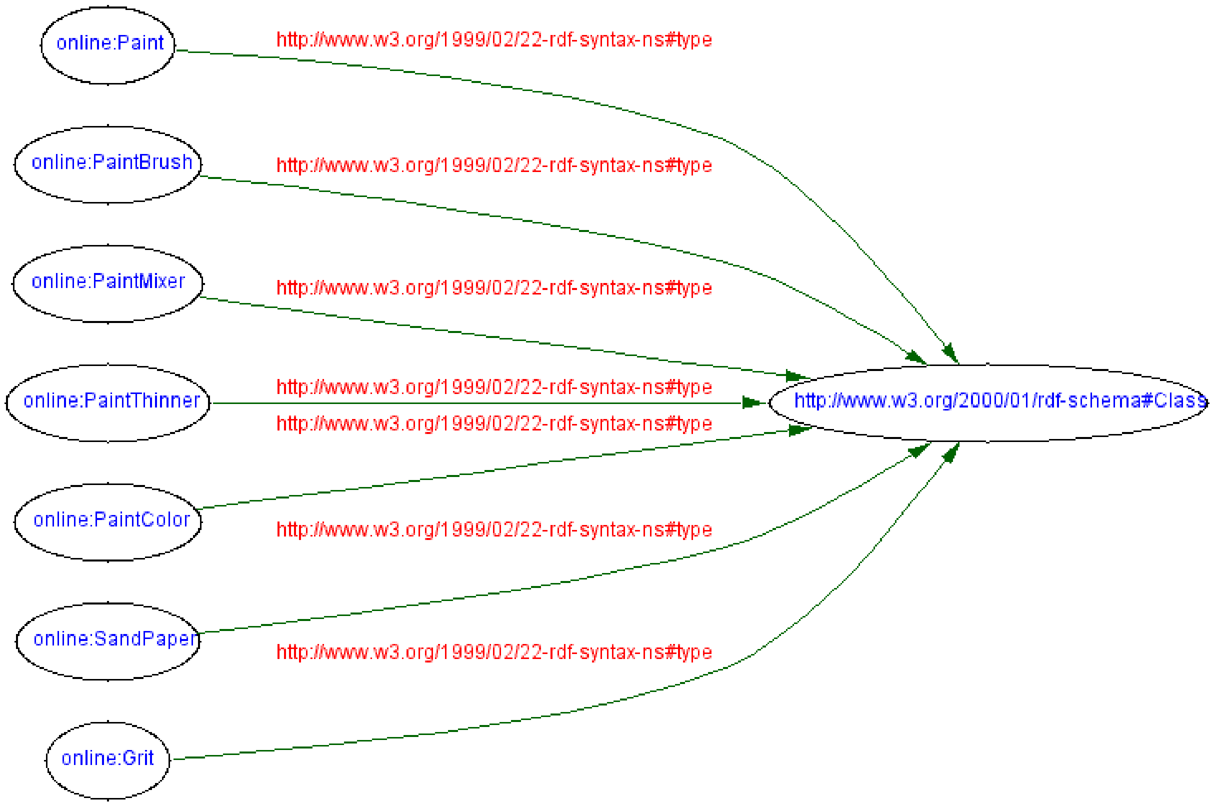
To illustrate the use of these guidelines in a single complete process description, consider a simple manufacturing scenario consisting of an activity “Finish product”. “Finish product” involves a “Paint” activity, followed by “Clean brush” and “Sand” activities, followed by another “Paint” activity, and concluding with final “Clean brush” and “Sand” activities. “Paint” has two sub-activities: “Mix paint” and “Apply paint”. “Clean brush” and can be done while waiting for the paint to dry. The first coat of paint is a primer coat, and the second coat uses the color blue. Sanding is performed the first time using 100 grit sand paper and the second time using 200 grit sand paper. Mixing is done using a paint mixer, and cleaning the brush is done using paint thinner.

This process when drawn as a UML activity diagram looks something like this:



Specifying Objects in RDF

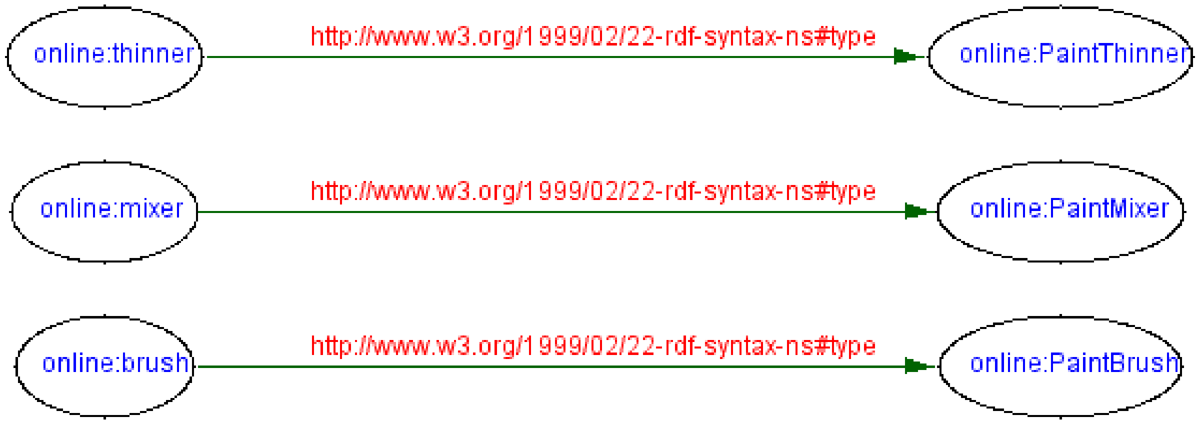
I begin by defining classes for the paint, brush, mixer, thinner, and sand paper objects. I also define classes for paint colors and sandpaper grit numbers. RDF [http://www.w3.org/RDF] is an XML-serializable language for expressing meta data for resources on the Web. The RDF data model is essentially a labeled directed graph, and an arc or node label may either be a URI or a literal. RDF Schema, defined using RDF, is a type system for use in RDF models. The following diagram shows the RDF graph.



The XML serialization (using the abbreviated syntax) is as follows:

```
<rdfs:Class rdf:ID="Paint"/>
<rdfs:Class rdf:ID="PaintBrush"/>
<rdfs:Class rdf:ID="PaintMixer"/>
<rdfs:Class rdf:ID="PaintThinner"/>
<rdfs:Class rdf:ID="PaintColor"/>
<rdfs:Class rdf:ID="SandPaper"/>
<rdfs:Class rdf:ID="Grit"/>
```

Next I specify instances of the brush, mixer, and thinner. The RDF graph appears as follows:

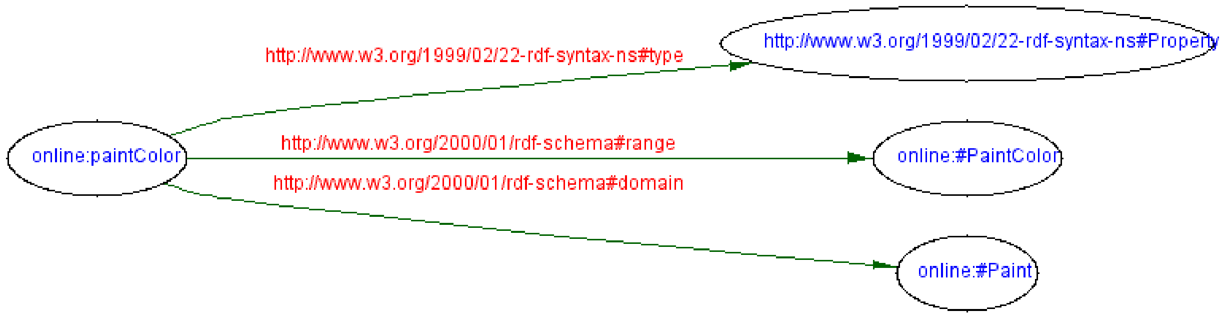

online:thinner
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
online:PaintThinner
online:mixer
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
online:PaintMixer
online:brush
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
online:PaintBrush

The XML serialization for these instances is:

```
<PaintBrush rdf:ID="brush"/>
<PaintMixer rdf:ID="mixer"/>
```

```
<PaintThinner rdf:ID="thinner"/>
```

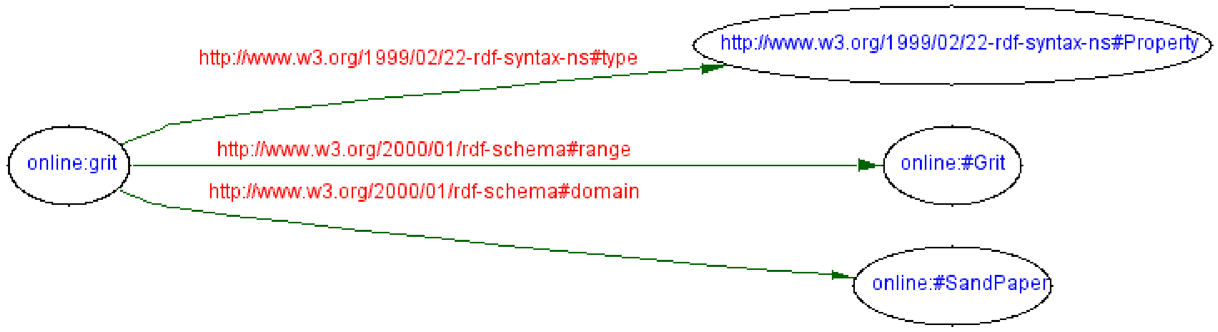
In order to specify instances of the paint and sand paper, I need to define some properties. I define a property paintColor whose domain is the `Paint` class and whose range is the `PaintColor` class. I use the convention of beginning class names with an upper case letter and beginning other names with a lower case letter. The RDF graph is:


http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.w3.org/1999/02/22-rdf-syntax-ns#Property
online:paintColor
http://www.w3.org/2000/01/rdf-schema#range
online:#PaintColor
http://www.w3.org/2000/01/rdf-schema#domain
online:#Paint

The XML serialization for this property is:

```
<rdf:Property ID="paintColor">
   <rdfs:range rdf:resource="#PaintColor"/>
   <rdfs:domain rdf:resource="#Paint"/>
</rdf:Property>
```

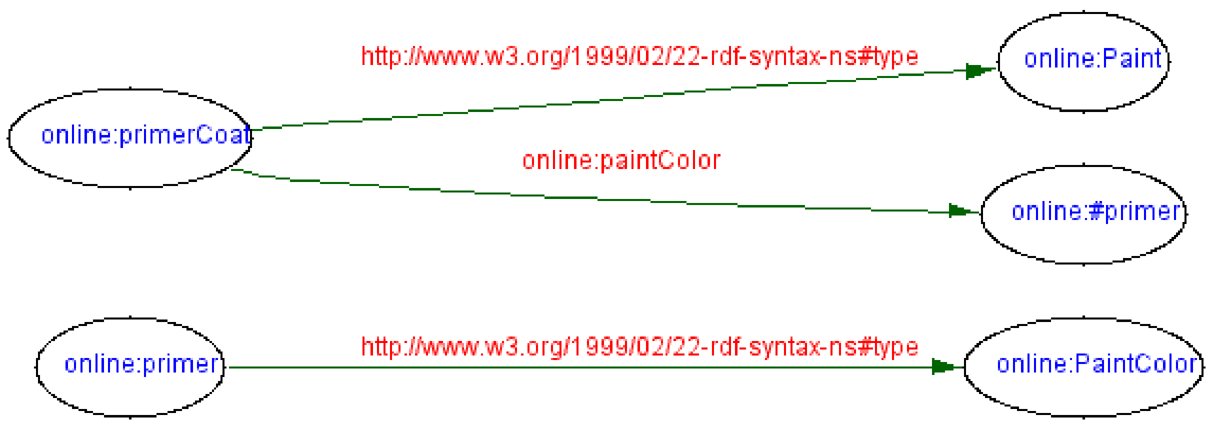
Similarly, I define a grit property with a domain of `SandPaper` and a range of `Grit`. The RDF graph is:


http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.w3.org/1999/02/22-rdf-syntax-ns#Property
online:grit
http://www.w3.org/2000/01/rdf-schema#range
online:#Grit
http://www.w3.org/2000/01/rdf-schema#domain
online:#SandPaper

The XML serialization is:

```
<rdf:Property ID="grit">
   <rdfs:range rdf:resource="#Grit"/>
   <rdfs:domain rdf:resource="#SandPaper"/>
</rdf:Property>
```

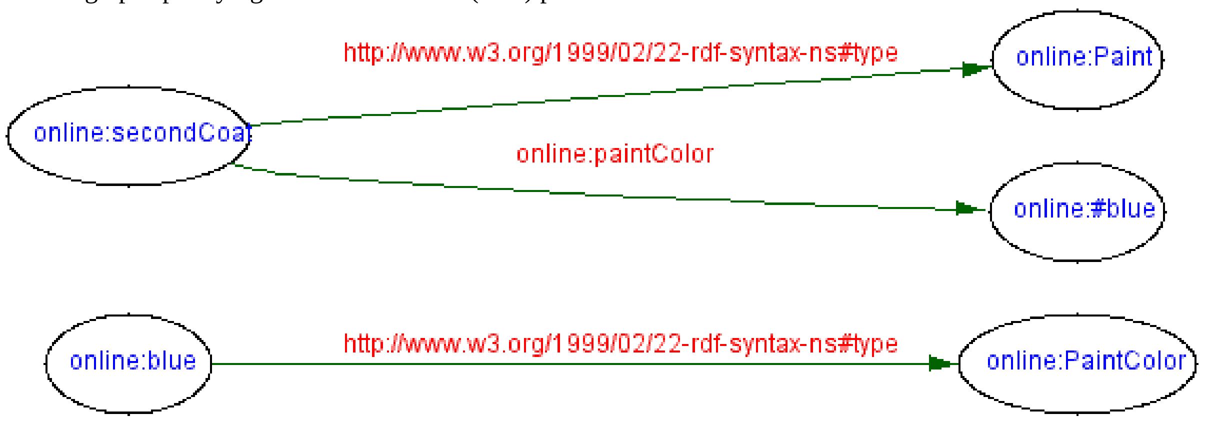
Now let us specify an instance for the primer coat of paint using the paintColor property I defined. The RDF graph is as follows:


http://www.w3.org/1999/02/22-rdf-syntax-ns#type
online:Paint
online:primerCoat
online:paintColor
online:#primer
online:primer
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
online:PaintColor

The XML serialization for the instance of the PaintColor property is:

```
<PaintColor rdf:ID="primer"/>
<Paint rdf:ID="primerCoat">
   <paintColor rdf:resource="#primer"/>
</Paint>
```

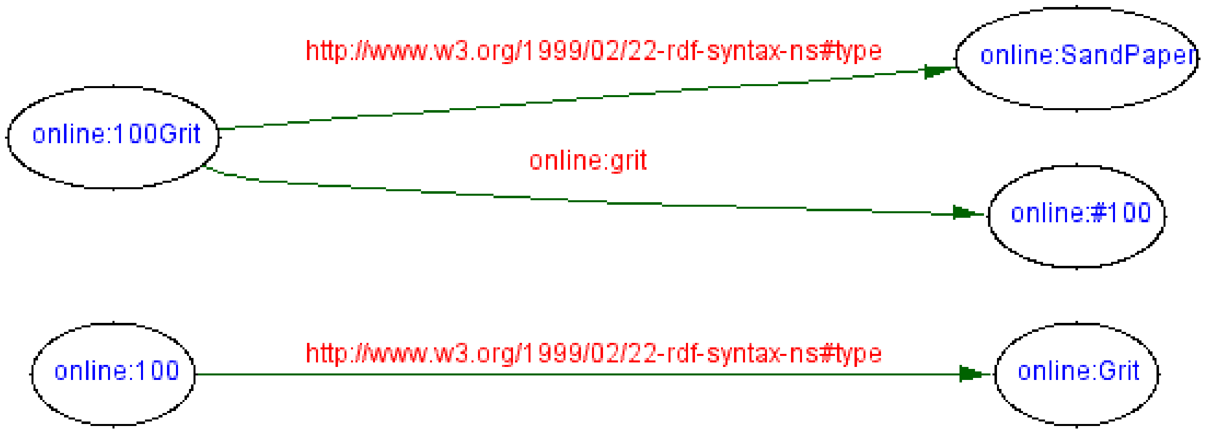
The RDF graph specifying the second coat of (blue) paint looks like:


http://www.w3.org/1999/02/22-rdf-syntax-ns#type
online:Paint
online:secondCoat
online:paintColor
online:#blue
online:blue
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
online:PaintColor

The corresponding XML serialization would be:

```
<PaintColor rdf:ID="blue"/>
<Paint rdf:ID="secondCoat">
   <paintColor rdf:resource="#blue"/>
</Paint>
```

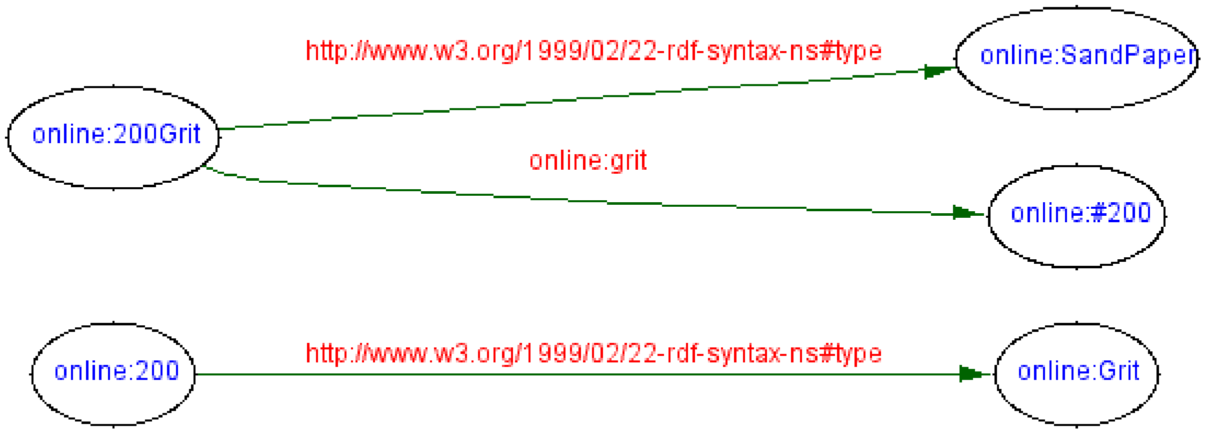
The RDF graph and XML serialization for the 100 grit sand paper instance are:


http://www.w3.org/1999/02/22-rdf-syntax-ns#type
online:SandPaper
online:100Grit
online:grit
online:#100
online:100
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
online:Grit

and:

```
<Grit rdf:ID="100"/>
<SandPaper rdf:ID="100Grit">
   <grit rdf:resource="#100"/>
</SandPaper>
```

Finally, the RDF graph for the 200 grit sand paper instance is:


http://www.w3.org/1999/02/22-rdf-syntax-ns#type
online:SandPaper
online:200Grit
online:grit
online:#200
online:200
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
online:Grit

The XML serialization will be:

```
<Grit rdf:ID="200"/>
<SandPaper rdf:ID="200Grit">
   <grit rdf:resource="#200"/>
</SandPaper>
```

The Process Description

I can now specify the process's time points, activities and activity occurrences.

Here is what the time points for our scenario might look like:

```
<timepoints>
   <timepoint id="t1">start</timepoint>
   <timepoint id="t2">done mixing paint</timepoint>
   <timepoint id="t3">done applying paint</timepoint>
   <timepoint id="t4">done cleaning brush and drying</timepoint>
   <timepoint id="t5">done sanding</timepoint>
   <timepoint id="t6">done mixing paint</timepoint>
   <timepoint id="t7">done applying paint</timepoint>
   <timepoint id="t8">done cleaning brush and drying</timepoint>
   <timepoint id="t9">done sanding</timepoint>
</timepoints>
```

Assuming the RDF representation of the resource objects is contained in a file with the relative URI `finish.rdf`, the “Finish product” activity from our scenario could be represented as follows:

```
<activity id="a1">
   <name>Finish product</name>
   <activity id="a2">
      <name>Paint</name>
      <activity id="a3">
         <name>Mix paint</name>
         <resource rdf:resource="finish.rdf#Paint"/>
         <resource rdf:resource="finish.rdf#PaintMixer"/>
      </activity>
      <activity id="a4">
         <name>Apply paint</name>
         <resource rdf:resource="finish.rdf#Paint"/>
         <resource rdf:resource="finish.rdf#PaintBrush"/>
      </activity>
   </activity>
   <activity id="a5">
      <name>Clean brush</name>
      <resource rdf:resource="finish.rdf#PaintBrush"/>
      <resource rdf:resource="finish.rdf#PaintThinner"/>
   </activity>
   <activity id="a6">
      <name>Dry</name>
   </activity>
   <activity id="a7">
      <name>Sand</name>
      <resource rdf:resource="finish.rdf#SandPaper"/>
   </activity>
</activity>
```

The XML representing the first occurrence of the “Paint” activity from our scenario might look like this:

```
<occurrence activity="a2" begin="t1" end="t3">
   <occurrence activity="a3" begin="t1" end="t2">
      <resource rdf:resource="finish.rdf#primerCoat"/>
      <resource rdf:resource="finish.rdf#mixer"/>
   </occurrence>
   <occurrence activity="a4" begin="t2" end="t3">
      <resource rdf:resource="finish.rdf#primerCoat"/>
      <resource rdf:resource="finish.rdf#brush"/>
   </occurrence>
</occurrence>
```

Next I have the first occurrences of the concurrent “Clean brush” and “Dry” activities.

```
<fork begin="t3" end="t4">
   <occurrence activity="a5">
      <resource rdf:resource="finish.rdf#brush"/>
      <resource rdf:resource="finish.rdf#thinner"/>
   </occurrence>
   <occurrence activity="a6"/>
</fork>
```

The next activity occurrence is the sanding of the primer coat with 100 grit paper. Then there are second occurrences of the "Paint", "Clean brush", "Dry", and "Sand" activities. Because the XML markup of these activity occurrences looks a lot like that of the activity occurrences already shown, I do not present it in this article.

Representation Issues

In the section called “XML Representation of Process Data”, I presented process representation guidelines that attempt to maximize XML's strengths while minimizing its weaknesses. However, XML developers should consider the following issues as well. There are no “one-size-fits-all” answers for these issues.

Late Versus Early Binding

In a late binding, the named components of the XML vocabulary do not directly correspond to specific constituents in the process. Instead they correspond to meta-concepts such as “activity”, “time point”, etc. The leftovers preparation and finishing process examples from the previous sections are both specified in a late-bound manner.

In an early binding, the named components of the XML vocabulary directly correspond to specific items. For example, an early-bound representation of the leftovers preparation activity occurrence from the section called Guide-“lines for Representing Process Data in XML” might look like this:

```
<prepareLeftovers xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
                  concurrentSubActivities="true">
   <reheatFood>
      <oven rdf:resource="#myMicrowave"/>
      <frozenMeal rdf:resource="#tunaCasserole"/>
   </reheatFood>
   <readNewspaper>
      <newspaper rdf:resource="#2001May29WashingtonPost"/>
   </readNewspaper>
</prepareLeftovers>
```

Because reheating the food and reading the newspaper can occur in parallel, the content model for `prepareLeftovers` should specify that `reheatFood` and `readNewspaper` can occur in any order.

Early bindings tend to be more succinct and human-readable than late bindings. However, each early binding requires its own DTD or schema. A late binding, on the other hand, allows for a single vocabulary to be used for multiple process descriptions. Thus, late bindings are best suited for applications involving a class of processes while early bindings may be best for situations involving a single process.

Choosing an XML Schema Language

I used DTD syntax in the examples from the section called “Guidelines for Representing Process Data in XML”. Alternatively, I could have used W3C XML Schema definitions, RELAX NG [http://www.oasis-open.org/committees/relax-ng/] patterns, or some other XML schema language. Although a DTD was adequate for specifying the vocabulary used in our examples, there are times when you need features that DTDs lack. For example, if you want to use multiple namespaces in your document (perhaps to combine your process vocabulary with another vocabulary), you will need a schema language that supports namespaces. You also might need more powerful datatyping.

The ability to define context-dependent element types can also be useful. For example, consider the content model for the `prepareLeftovers` element in our early binding example specifying that its sub-activities can occur in any order. This could be represented using DTD syntax as

```
<!ELEMENT prepareLeftovers
          ((reheatFood, readNewspaper) | (readNewspaper, reheatFood))>
```

Although this isn't too bad for only two sub-activities, imagine how clumsy it would be to use DTD syntax to specify that ten sub-activities can occur in any order. Clearly, W3C XML Schema (using `all`) or RELAX NG (using `interleave`) would be better suited for representing arbitrarily large collections of unordered elements.

XLink Versus ID/IDREF

In our examples, I used attributes of type ID to uniquely identify time points and activities. Activity occurrences used attributes of type IDREF to reference their beginning time point, ending time point, and associated activity. Alternatively, I could have used XML linking (XLinks) [http://www.w3.org/XML/Linking] for cross-references. XLinks are more powerful than IDREF in that they allow references to things outside the XML document, permit us to associate semantics with links, and support aggregation of links. On the other hand, unlike ID/IDREF, XLink applications don't detect dangling cross-references.

While ID/IDREF may be best for process descriptions contained in a single document, XLink may be best for process descriptions spread across multiple documents, lightweight applications where no DTD or schema is available for the process description, or applications with sophisticated cross referencing requirements.

Hierarchical Versus Flat Descriptions

Although our XML representation guidelines suggest using hierarchies of elements to represent activities and activity occurrences, sometimes this is not practical. For example, if multiple activities share a common sub-activity, a hierarchical representation would require that the same information be repeated multiple times. Also, processes with complex timing relationships between activities (e.g. activities that overlap but have different starting and/or ending times) might be difficult to represent in a hierarchical form.

An alternative to using hierarchies is to represent containment relationships using a "relationship" element whose content is one or more cross-references. For example, suppose I wanted to represent our early binding example such that the `reheatFood` and `readNewspaper` elements are not children of `prepareLeftovers`. I could do this using a `subactivities` element to specify the relationship, `reheatFood-ref` to cross-reference `reheatFood`, and `readNewspaper-ref` to cross-reference `readNewspaper`. The result would look like this:

```
<process xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <prepareLeftovers concurrentSubActivities="true">
      <subactivities>
         <reheatFood-ref ref="rf1"/>
         <readNewspaper-ref ref="rn1"/>
      </subactivities>
   </prepareLeftovers>
   <reheatFood id="rf1">
      <oven rdf:resource="#myMicrowave"/>
      <frozenMeal rdf:resource="#tunaCasserole"/>
   </reheatFood>
   <readNewspaper id="rn1">
      <newspaper rdf:resource="#2001May29WashingtonPost"/>
   </readNewspaper>
</process>
```

Alternative Process Representation Approaches

I have used the concepts defined in PSL as the foundation for ideas for XML representation of processes. Two other sources of process representation semantics that may be worth considering are the UML and the Workflow Management Coalition's (WfMC) Workflow Reference Model.

UML

The UML, an Object Management Group (OMG™) [http://www.omg.org] standard, is actually a family of graphically oriented modeling languages intended to be used for software design. Just as XML has become the dominant language for serializing and transporting data, UML has become the dominant modeling language. Unfortunately for process modelers, no single type of UML diagram captures all of the information needed to describe a process. UML activity diagrams (see the figure at the beginning of the section "Extended Example: A Finishing Process" for an example) do a good job modeling complicated sequences and parallelism. However, activity diagrams are not the best choice for representing the relationships between activities and objects. UML interaction diagrams do a much better job describing how actions and objects collaborate.

The OMG's XML Meta Data Interchange (XMI®) standard provides a way to represent UML models in XML. Thus it is possible (although probably not easy) to obtain an XML representation of a process by modeling it both as an activity diagram and as an interaction diagram, generating XMI from the two diagrams, and using XML Style Language Transformations (XSLT) [http://www.w3.org/Style/XSL/] to convert the exported XMI into a single XML process description.

The WfMC's Workflow Reference Model

The WfMC [http://www.wfmc.org/] is consortium of workflow vendors, users, and researchers developing interoperability and connectivity standards for workflow products. The WfMC's Workflow Reference Model is a specification identifying the characteristics, terminology and components of workflow management systems. The Workflow Reference Model provides the context for other WfMC specifications.

One such WfMC specification is the XML Process Definition Language (XPDL), a format for the exchange of workflow process definitions. The XPDL DTD defines an XML syntax for workflow-related concepts such as state transitions, activities, participants, etc. Although XPDL is (at the time of this writing) in “Draft” status, it is further along than PSL in the sense that a specific XML vocabulary has already been specified. For representing process descriptions involving workflow, XPDL may well be worth a look.

Summary

In this article, I discussed the design of XML vocabularies for describing discrete processes. Taking PSL concepts as a starting point, I established some guidelines and demonstrated their use with examples. I then enumerated some additional design issues and additional sources of process semantics.

For PSL or any other process specification language to achieve widespread success, there needs to be some sort of killer “killer app”. This killer app, if and when it emerges, is certain to use XML for exchanging process descriptions. The application could be a Web-based process visualization tool, perhaps employing the W3C's Scalable Vector Graphics (SVG) [http://www.w3.org/Graphics/SVG] standard. Or maybe it could be a framework for distributed workflow management that makes workflow tools affordable for the masses. Only time will tell.

Resources

Knowledge Interchange Format (KIF)	http://logic.stanford.edu/kif
Object Management Group	http://www.omg.org
Process Specification Language	http://www.nist.gov/psl
PSL Ontology	http://www.nist.gov/psl/psl-ontology
Workflow Management Coalition	http://www.wfmc.org
World Wide Web Consortium (W3C) - developers of XML, RDF, and related standards	http://www.w3.org