CAPE VISIONS
Software To Simplify Complexity

Rough Draft

A Comparison of XPDL, BPML and BPEL4WS

Cape Visions

Robert Shapiro

Version	Date	Author	Description
1.0	3/25/02	RMS	First draft of Comparison
1.1	4/2/02	RMS	Revision based on suggestions by Roberta Norin, APEngines.
1.2	4/3/02	RMS	Revision after WfMC WG1 teleconference
1.3	4/15/02	RMS	Minor edit
1.4	8/27/02	RMS	Revision after BPML 1.0 (6/24/02) release, WfMC XPDL 1.0 beta (7/30/02) and BPEL4WS 1.0 (8/9/02).

1 Introduction

The Business Process Modeling Language (BPML) is representative of a new family of process definition languages intended for expressing abstract and executable processes that address all aspects of enterprise business processes, including in particular those areas important for web-based services. Microsoft's XLANG is another member of this family, as is IBM's Web Services Flow Language (WSFL). These latter two have now been combined in BPEL4WS.

In this paper we focus on a comparison of BPML with XPDL, the WfMC proposed standard for an XML-based process definition interchange language. Comments (in red) have been added to extend the comparison to BPEL4WS, hereafter abbreviated to BPEL.

Section 2.11.4 (Activity Details) contains brief descriptions included for the benefit of readers unfamiliar with BPML.

1.1 Objectives

Our primary objective is to clarify the differences between the BPML and XPDL (and BPEL) paradigms. We are interested in exposing what can be done with one language and cannot be done, or done only with difficulty in the other. When simple extensions are possible, we propose them.

We are also concerned about the work being done by the three standards organizations:

- WfMC
- OMG
- BPMI

This effort intends to create a common modeling notation (BPMN). The comparison of BPML and XPDL should expose some of the challenges in this undertaking.

As of August, 2002, BPMN remains a BPMI activity: no significant cooperation has developed between the three organizations

In addition to BPML, on 26 June 2002, BEA Systems, Intalio, SAP and Sun announced the publication of the XML-based WSCI, a specification that:

- Defines the behavior of Web service interfaces
- Forms part of the ongoing Web service process flow composition efforts (often referred to as orchestration, choreography or workflow)

Incorporating many aspects of BPML, WSCI focuses on the choreography of web services. We do not explore this topic further in this paper.

There are at least three other important projects that address Business Process Management System (BPMS) issues in a Web Services context:

- ebXML(OASIS/UN)
- WSFL (IBM)
- XLang (MS)

In August 2002 Microsoft/IBM released a new specification, BPEL4WS, which combines XLANG and WSFL.

It is our intent to include a discussion of functionality contained in any of these which should be part of a business process definition language, BPMN or supported by BPM simulation technology.

2 BPML/XPDL Comparison

2.1 Overview

BPML and XPDL (and BPEL) are XML-based process definition languages. They provide a formal model for expressing executable processes that addresses all aspects of enterprise business processes. They are based on significantly different paradigms.

Each paradigm utilizes **activities** as the basic components of process definition. In each, activities are always part of some particular **process**. Each has instance-relevant data, **property** for BPML and **workflow-relevant data (data fields)** for XPDL (and Containers for BPEL), which can be referred to in routing logic and expressions.

BPML is conceived of as a block-structured programming language. Recursive block structure plays a significant role in scoping issues that are relevant for declarations, definitions and process execution. Flow control (routing) is handled entirely by block structure concepts (e.g. execute **all** the activities in the block sequentially).

XPDL is conceived of as a graph-structured language with additional concepts to handle blocks. Scoping issues are relevant at the package and process levels. Process definitions cannot be nested. Routing is handled by specification of transitions between activities. The activities in a process can be thought of as the nodes of a directed graph, with the transitions being the edges. Conditions associated with the transitions determine at execution time which activity or activities should be executed next

BPEL is a block-structured programming language, allowing recursive blocks but restricting definitions and declarations to the top level. Within a block graph-structured flow concepts are supported to a limited extent, constrained by inheritance from previous generation workflow software (only acyclic graphs, hence no loops; some constraints on going across block boundaries; a complicated semantics for determining whether an activity actually happens).

BPML focuses on issues important in defining web services. This is reflected in several ways:

- Activity types specifically for message interchange, event handling, compensation (in case of failure), delay.
- Attributes to support instance correlation, extraction of parts of messages, locating service instances.
- Support for transactions, utilizing the block structure context, exception handling and compensation.

XPDL focuses on issues relevant to the distribution of work.

- Activity attribute specifies the resource(s) required to perform an activity. This is an expression, evaluated at execution time, which determines the resource required.
- Activity attribute specifies the application(s) required to implement an activity.

- These concepts together support the notion of a resource (e.g. participant), in conjunction with an application, performing the activity. The implementation of work list handlers to achieve this lies outside the domain of the process definitions.

BPEL focuses on issues important in defining web services and does this in a way which is quite similar to BPML.

2.2 Block Structured versus Directed Graph

It is not the purpose of this paper to argue the merits of these two approaches. We make two points in this regard.

- Block structures work well in programming languages.
- Business operations people are used to flow diagrams and other graphical notations.

It seems likely that business users of BPML and XPDL would prefer to use a GUI based, at least in part, on diagrams. We are concerned here with questions about what can be represented in one language and not the other.

Translation of blocked-structured flow control (routing) into a graph structure presents no fundamental difficulties. The reverse is more problematic. This can be facilitated by imposing a set of restrictions on the graph structure that guarantee it to be ‘well-structured’. It is likely that Business Process Management Systems that use BPML will support some type of graphical tool for process definition that imposes such restrictions. It remains to be seen whether such restrictions limit the usability of BPML. We do not pursue this topic in this paper.

BPEL attempts to offer the best of both approaches by introducing a **flow** construct and using **links** to create ‘arbitrary’ flow dependencies between the activities contained within the flow construct. However, there are constraints which rule out loops and crossing certain structural boundaries; additionally the semantics relies on a complicated formulation which tests and propagates the status of links. It is possible that a graphical front end could simplify this and be more user-friendly to the business analyst.

2.3 Definitions and Recursive Block Structure

BPML makes extensive use of block structure scoping related to definitions and declarations. Complex activities refer to activity sets which have an associated context. In the context it is possible to declare or re-declare properties, define or re-define processes (nested processes) and so forth. (Many other features are scoped by the context in which they appear, including error handling, transactions and connectors (for message handling). Since a complex activity can appear in an activity set, this nesting is recursive.

XPDL only allows process definitions on the top level. Hence there are no nested processes. Since workflow relevant data is declared either on the top level, or within a process definition, it is limited to 2 scope levels. We make no assessment here as to whether nested process definitions are an important feature. BPEL does not support nested process definition. Furthermore, the equivalent of workflow relevant data, **containers**, are global in scope. (The current spec also fails to make clear whether (or how) containers are instance-specific).

In what follows we make use of two new constructs included in XPDL 1.0 beta, **blockactivity** and **activityset**. The block activity is then like the complex activities in BPML, with attributes for designating the type of complex activity and other information appropriate to defining a context for an activity set. Including data field declarations in the context would allow the same scoping possibilities as BPML. To implement nested processes, process definitions would also have to be included.

Most of the features associated with BPML complex activities could be represented in XPDL by the block activity. This construct refers by name to a set of activities that have no transitions outside the set. The construction process for the BPML **all** activity, for instance, would simply introduce a first and last activity within the activity set, where the first activity is an **andsplit** and the last activity is an **andjoin**. There would be a transition from the first activity to each of the activities within the block and a transition from each activity to the last activity. Other types of complex activities could be represented by transitions that implement the appropriate control logic. In the sequel we suggest an alternate approach which makes the translation from BPML to XPDL easier, but passes some of the burden onto the workflow or simulation engine that executes the XML definitions. In so doing we are not making a recommendation to change XPDL.

(The August 2002 WfMC XPDL 1.0 beta has included a version of **BlockActivity** and **ActivitySet** that can by trivially extended to implement complex activities.)

2.4 Specialized Atomic Activities

BPML (and BPEL) includes a number of specialized atomic activities. Some of these in turn require a particular set of attributes to support their specific function. In XPDL there are several basic types:

- Implemented by sub-flow, synchronous and asynchronous.
- Implemented by application
- Routing activity (dummy, for routing purposes only)
- Block activity (proposal: replaces loop activity and inline block)

It is natural to map the atomic BPML activities into XPDL activities. There are several issues.

- BPML (and BPEL) does not have the notion of an application. The BPML Call activity is identical (except for scoping issues) to an XPDL activity implemented by a synchronous sub-flow. The BPML Spawn activity is like the XPDL asynchronous sub-flow, except for some special bookkeeping aspects.
- In BPEL all processes are instantiated by reception of a message. ("The only way to instantiate a business process in BPEL4WS is to annotate a `receive` activity with the `createInstance` attribute set to `"yes"`").
- Otherwise, additional attributes must be used in XPDL to carry the information needed by the specialized BPML activity.

2.5 Activities and Attributes for WSDL Messages

BPML (and BPEL) build on top of WSDL. They each have specific activities with the appropriate attributes to utilize the standard WSDL messages. XPDL activities would be extended to do the same by the addition of appropriate attributes.

BPML	**BPEL**	**XPDL**
action Implements the standard WSDL message patterns: One-way, request-response, solicit-response, notification	receive, reply, invoke	Would required additional attributes (and applications/tools and library functions)
correlation Used to match the message to the right instantiation of the process.	correlation	Workflow relevant data is instance specific. The correlation attribute would have to be used to identify the right instance.
locator Used to find the correct service. Refer to WSCI spec.	Service link, partners, service reference	
call An action can perform an arbitrary set of activities only if its semantics require that these activities be performed in order for the action to complete, specifically when performing the WSDL request-response operation.	Not supported	
output, selector Used to construct messages from property values.	propertyalias, assign, query, container	
Connector Used in the specification of the interaction between services in the Global Model. Refer also to WSCI spec.	See Service Composition, also Relationship to WS-Transaction Specification	

2.6 Web Services Orchestration

2.6.1 BPML and WSCI

2.6.2 BPEL and Business Protocols

Business processes can be described in two ways. *Executable* business processes model actual behavior of a participant in a business interaction. *Business protocols*, in contrast, use process descriptions that specify the mutually visible message exchange behavior of each of the parties involved in the protocol, without revealing their internal behavior. The process descriptions for business protocols are called *abstract processes*. BPEL4WS is meant to be used to model the behavior of both executable and abstract processes.

2.7 Specialized Complex Activities

Complex BPML activities refer to sets of activities and provide some type of routing logic or flow for the entire set.

XPDL 1.0 includes a **block** activity which refers to an activityset. This block activity could have an attribute that specifies the flow logic for the activities in the activity set, or the flow logic could be generated and represented by transitions between the activities in the set.

Additional information, such as **context**, would be provided by attributes of the block activity. The activity set could be defined with a block name and referred to by the block activity. This would make the activity set definition reusable. On the other hand, to match BPML scoping, the activity set should be defined within the block activity. In this solution there need be no transitions defined for any of these activities, since the routing logic is completely specified by the block activity.

2.8 Transactions and Exception handlers

BPML (and BPEL) provide constructs for supporting **Transactions** and handling various types of errors or exception. This includes the so-called ACID or atomic transactions, as well as open nested transactions (supported in BPEL as Long **R**unning **T**ransactions). The constructs include various attributes, scoping rules and error handling logic. They could map into XPDL attributes or special library functions. No discussion here of details. Some of these require a sophisticated syntax which should be handled by appropriate extension of the XPDL XML Schema. In BPML (and BPEL) these all have a well-defined semantics.

BPML	**BPEL**	**XPDL**
compensate Associated with an **exception handler** in a **context.**	compensate Associated with a **compensation handler** in a **scope**.	Would required additional attributes and library functions
event handlers time-out, fault	event handlers compensation handler. fault handler (Used in	XPDL 1.0 has been extended to include the specification of Deadlines

	conjunction with **catch** and **throw**.)	and an exception handler for deadlines and other exception conditions.

2.9 XPDL Constructs with no BPML (or BPEL) Analog

We have already discussed block structure versus graph structure in respect to flow logic and do not repeat that discussion here. An arbitrary XPDL activity network cannot be represented directly in BPML (or BPEL). One which satisfies certain constraints can be.

- BPML (or BPEL) provides no way of declaring participants or defining, in the activities, an expression, possibly based on the instance values of properties, that specifies the resource(s) required to perform the activity. Of course it would be easy to add this construct to the definition of an activity, but it is not immediately clear how the semantics would be defined.
- BPML (or BPEL) does not have the concept of application, neither in a declarative context for the package, nor as an implementation of an activity. Another way of looking at this is to say that there is no distinction being made between a BPML process and an application invoked in performing an activity.
- Both of these constructs could be supported by utilization of the BPML **action** activity. For instance, using the WSDL **Solicit-response,** a message containing the performer expression (or its value, as a set of participants) could be sent to a service which implements the work list handler and responds with the assigned participant(s). This could be extended to include designation of the application.
- XPDL contains a number of special attributes useful in areas such as version control, simulation and so forth. These could easily be added to BPML (or BPEL).

2.10 High level constructs

BPML	**XPDL**	**BPEL**
package	Package	No equivalent
process Instantiation issues must be examined. Top level processes are special. Processes may be instantiated by message arrival. Processes are the only reusable	WorkflowProcess In addition to input/output parameters, other required attributes include: **instantiation type** and **scope**. Process contains an **activitySet**.	process

units.		
Nested process Definition within a context (such as a top level process).	Would require **block** activity with an element that allowed definitions, including process definitions.	No nested processes.
import	ExternalPackage	import
activitySet A **context** is associated with an activity set. Properties are shared within an instance of a context. Scoping issues to be clarified.	activityset (introduced in XPDL 1.0 beta) This is the only construct for grouping activities other than process. Scoping issues remain. Used in 1.0 beta to implement sub-map.	Scope is used in conjunction with compensation handlers and fault handlers
complex activity Consists of one or more activity sets. Are activity sets reusable? I don't think so. All, choice, foreach, sequence, switch, until, while.	block activity. Refers to activityset by name. **Proposal: all BPML complex activities are xpdl block activities referring to an activityset and using other attributes to describe routing logic (or adding transitions and conditions to the activities in the set).**	Structured activities Sequence, switch, while, pick and flow.
activity There are a lot of different kinds of atomic activities and complex activities.	activity It seems natural to use activity attributes to represent the different kinds of activities.	activity
See separate tables for atomic activities and complex activities.	Atomic activities map to xpdl activities. Other attributes used for details. Complex activities map to xpdl route activities with a reference to the appropriate **block**. Other attributes used for control logic.	

2.11 Activities

2.11.1 Activity Set and Complex Activities

BPML	**XPDL** For all these cases: a **block** activity referring to an **activity set** by name. All flow logic could be made explicit (implemented by transitions and conditions) or retained in the block activity**.**	**BPEL**
activitySet	Attributes to specify **activitySet** and **context** information	scope
context Declarative information associated with an activitySet.	Declarations, including **properties** which are like **workflow relevant data** (e.g. data fields or variables)	Associated with scope
property See discussion in table about context.	Workflow relevant data Scoping differences	container
Selector Obtains a **property value** from a **message**.		**part** and **query** attributes, used in extracting property values from messages (and containers)
all	Attribute to specify **all**. A special case of andsplit/andjoin. See also spawn/join.	flow See discussion of links, join conditions and graph structure.
choice	Attributes to specify **choice** and **event handlers**	pick
foreach	Attribute to specify **foreach** and **list** (as expression)	No equivalent
sequence	Attribute to specify **sequence**.	sequence
switch	Attribute to specify **switch** and list of (**condition,activitySet**) pairs.	switch
until	Attribute to specify **until** and **condition**	No equivalent
while	Attribute to specify **while** and **condition**	while

2.11.2 Atomic Activities

BPML	**XPDL**	**BPEL**
action performs an indirect assignment as the result of any operation that receives an input message. By default the entire message contents are assigned to a property with the same name as the message. Handles all WSDL message patterns.	Attributes for **correlate** (list), **locate**, **call**, **portType** and **operation**.	**receive** **reply** **invoke** These handle the various WSDL message patterns. No equivalent to **call**.
assign Changes the value of a property.	Attributes for **property** and **expression**	**assign** Changes the value of an XML value(message) in a container
call	Implemented by synchronous sub-flow	No equivalent. All instantiations done by receiving a message.
compensate	Attributes for **transaction**.	compensate
delay	Attribute for specification of delay	wait
empty	Dummy activity (route).	empty
fault	Attribute for name of fault	throw/catch
join This waits for n instances of a process to complete. Context of instantiation relevant. Used with **spawn**.	Attributes to specify **process** and **count**.	No equivalent. **Join** condition is used in conjunction with **flow** construct and **links** to handle concurrency and graph structures.
spawn Used with **join**.	Implemented by asynchronous sub-flow. However, a major use of spawn is in connection with join, and this is a special kind of **andsplit** followed by **andjoin**. Bookkeeping entirely automatic.	All instantiations done by receiving a message.
No explicit terminate	No explicit terminate	terminate

2.11.3 Context related constructs

BPML	XPDL
property Inherited from all ancestors (block structure allows arbitrary depth). Declaration makes it local to context in which it occurs. Package level property declarations are instantiated in each top level process and thereafter the process instance property values are independent	Workflow relevant data Only two levels of declaration: package and process. Route activity for block (activitySET) could allow more declarative levels.
Instance properties Whenever an activity, transaction or process is instantiated, a property is instantiated in the current context to provide the instance identifier and state of that activity, transaction or process. These properties are collectively known as instance properties.	Requires implicit declaration of instance properties. Used in exception handling, etc.
exception Same scoping idea.	Need to add exception information to process definition and new block construct (routing block for activity set).
transaction Similar issues as exception	Same treatment as exception
connector Similar issues as exception	Same treatment as exception
completion activitySet	
process definition (nested)	Process definitions can't be nested. What problems arise if this is changed? Could the activity set block context information simply allow a process definition? What are the run time and name scoping issues that must be dealt with?
Event handlers associated with context Invoke activitySet Message, time-out, fault all are events	

2.11.4 Activity Details

2.11.4.1 Action

Action provides the context for performing an operation. In particular, it pertains to operations involving the exchange of messages with participants.

```
<action
      name = NCName
      portType = QName
      operation = NCName
      {extension attribute}>
      Content: (documentation?, correlate*, locate?, call?,
      output*)
</action>
```

An action does not define the operation that is to be performed, but indicates which operation will be performed and provides the execution context. An action is atomic and so can only refer to a single atomic operation.
Operations are defined by other specifications and imported by a BPML document using the *import* element. Referencing operations defined by the WSDL specification is a normative part of the BPML specification

Correlate

Correlating an action establishes a relation between the context in which the action occurs and the message received by the action through properties that are shared by the context and the message.

Locate

A locator is required if the action must identify the service instance. This specifically applies when performing the WSDL *notification* and *solicit-response* operations. The *locate* element is not allowed for other WSDL operations. A service instance can be located in one of three ways:

1. dynamically by URI
2. dynamically by lookup
3. statically

. Call a process. Like call activity

Output

It is necessary to construct to outputs only for actions that involve sending a message. This applies specifically when performing the WSDL request-response, notification and solicit-response operations. The output element is not allowed for the WSDL one-way operation.

.WSDL operations

When the action performs an operation defined by WSDL, the **portType** and **operation** attributes are used. The *portType* attribute references the WSDL port type definition, while the *operation* attribute references the particular operation of that port type definition. Actions may refer to the following WSDL operations:
• **One-way** The process receives the input message. Correlation may be required.
• **Request-response** The process receives the input message, constructs and sends an output message back to the sender. Any work done between the input

and output messages is performed by calling a process. Correlation may be required.
• **Solicit-response** The process constructs and sends a message and waits for a response from
the recipient. The recipient must be unambiguously identified by using the *locate* element.
• **Notification** The process constructs and sends a message. The recipient must be unambiguously identified by using the *locate* element.

•

2.11.4.2 All

Activities are executed in non-sequential order. A particular order must not be enforced, however, there is no requirement for activities to be executed in parallel.

2.11.4.3 Assign

The *property* attribute provides the property name.
The value is constructed using one of the following three means:
• **value** Provides an XML value that is statically provided in the content of that element
• **select** Provides an XPath expression that is evaluated in the context in which the activity is used
• ***extension element*** Supports other mechanisms by which the value is constructed
The three uses are mutually exclusive and cannot be combined in the same element. If the extension element defines a form of expression, such as an XQueryX query, it is always evaluated in the context in which this activity is used.

2.11.4.4 Call

• Can instantiate processes whose definition is visible from the current context. The process is instantiated in the same context in which it is defined, which may be different than the context from which the process is called.
• Waits until the instantiated process completes, either successfully or with a fault. If the called process faults, the *call* activity completes with the same fault code.
• Does not directly affect any *call*, *spawn* or *join* activity relating to the same process and occurring in the same or different context.

2.11.4.5 Choice

The **choice** activity is a complex activity. It selects and executes one activity set in response to a triggered event.

2.11.4.6 Compensate

The *compensate* activity is an atomic activity. It performs compensation for all instances of the named transaction.

2.11.4.7 Delay

The *delay* activity is an atomic activity. It expresses the passage of time.

2.11.4.8 Empty

This activity can be used in places where an activity is expected, but no work is to be performed.

2.11.4.9 Fault

The fault code is specified using the *code* attribute. The fault occurs immediately in the current context, see the definition of exception handling for how faults and other exceptions are handled.

2.11.4.10 Foreach

The *foreach* activity repeats once for each item in the resulting list, in the same order in which the list was constructed. The value of the current item is held in the property *bpml:current*. That property is accessible only from the context of the activity set.

2.11.4.11 Join

The *join* activity is an atomic activity. It waits for instances of process to complete.

2.11.4.12 Sequence

The *sequence* activity is a complex activity. It performs all the activities within the activity set in sequential order.

2.11.4.13 Spawn

The *process* attribute names the spawned process and does not wait for it to perform any activity. Instead, the activity completes immediately. The *spawn* activity can only instantiate processes whose definition is visible from the current context. The process is instantiated in the same context in which it is defined. This context may be different than the one from which the process is spawned.
This activity modifies the process instance list. This list is maintained as a property and has the same name as the process under the current context. As such, it can affect *join* activities.

2.11.4.14 Switch

The *switch* activity is a complex activity. It selects and executes one activity set based on the evaluation of one or more conditions.

2.11.4.15 Until

The activity set is executed at least once. After completion of the activity set, the condition is evaluated. The process is repeated if the condition evaluates to *false*. Otherwise, the *until* activity completes.

2.11.4.16 While

The condition is evaluated once before the activity set is executed. The activity set is executed only if the condition evaluates to *true*, otherwise the *while* activity completes. This process is repeated until the condition evaluates to *false*.

3 Questions

- In BPML **activity sets** are not re-usable. Implementing them as re-usable inline blocks might depend upon whether they have a context specified or not. Need to examine scoping issues
- A process refers to an activity set. A process can evidently not be defined within an activity set. So how do you define a nested process? Consider the following:
    - Although a process definition is based on the activity type, **a process definition cannot be used as an activity within an activity set**. An activity must execute within a context. As a result, activities must be part of an activity set. In turn, activity sets are always contained within a larger definition. This definition can be a complex activity, a process definition, or some other construct such as exception or compensation.
    - A process is an activity set that is not contained within any other activity.
    - Notice that a **context allows definition of a nested process**: element defines a nested process that will be instantiated in that context and overrides any other definition with the same name that would be visible in this context.
    - A complex action may involve the execution of a nested process, by instantiating that process and waiting for it to complete within the confines of the action. This can be done using the **call**. When referring to a WSDL operation this element is used only for request-response operations.

4 References

[1] BPML working draft March 25, 2002.

[2] BPML working draft June 24, 2002.

[3] BPEL4WS (Business Process Execution Language for Web Services) Version 1.0 Aug 9, 2002

[4] Workflow Process Definition Language - XML Process Definition Language Document Number WFMC-TC-1025 Document Status – Draft 0.04a (Alpha Status) March 01, 2001 Version 0.04 (Draft)

[5] Workflow Process Definition Interface -- XML Process Definition Language. Document Number WFMC-TC-1025 Document Status - XPDL 1.0 beta. (July 30, 2002) .

[6] WSDL Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001

[7] Web Services Choreography Interface (WSCI) 1.0, BEA, Intalio, Sun, SAP et al, June 2002

[8] Web Services Coordination (WS-Coordination) BEA Systems, International Business Machines Corporation, Microsoft Corporation 9 August 2002