

# Methodology and Technology Services

| [Home](#) | [Courses](#) | [Certification](#) | [Projects](#) | [Papers](#) | [Online Store](#) | [Contact Us](#) |

## THE ENTERPRISE NEWSLETTER

Issue 27:

# SERVICE-ORIENTED ARCHITECTURE (SOA)

[Printable PDF Version](#)

## CONTENTS

- [Service-Oriented Architecture](#)
  - [The Importance of Service-Oriented Architecture](#)
  - [Introduction to Service-Oriented and Event-Driven Architectures](#)
  - [Business Process Execution Language \(BPEL\)](#)
  - [Business Process Modeling Language \(BPML\)](#)
  - [ebXML Business Process Specification Schema \(BPSS\)](#)
  
- [Previous Issues of TEN](#)
- [Glossary of Terms](#)
- [New Subscribers to TEN](#)

---

PERTH, AUSTRALIA – October 11, 2004: I have previously discussed Enterprise Architecture methods, as well as technologies for rapid delivery into production of priority business processes. These technologies have included XML, Enterprise Portals and Web Services. In this issue I will cover the concepts of rapid delivery technologies based on Service-Oriented Architecture (SOA) and Business Process Management (BPM). I will discuss a number of XML-based BPM languages: BPEL4WS (Business Process Execution Language for Web Services); Business Process Modeling Language (BPML); and Business Process Specification Schema (BPSS) for ebXML.

Clive Finkelstein  
TEN - The Enterprise Newsletter

[Back to Contents.](#)

---

## SERVICE-ORIENTED ARCHITECTURE

Web Services technology has now advanced so that functions within existing application programs and suites – as well as functions within ERP, CRM, SCM and other packages – can be easily and reliably published to the Intranet or Internet for remote execution using SOAP, WSDL and UDDI. But what has been missing until now is an automated way to invoke available Web Services based on Business Rules. This technology is now becoming available with Service-Oriented Architecture using Business Process Management (BPM) languages and tools.

### The Importance of Service–Oriented Architecture

The term: “Service-Oriented Architecture” (SOA) up until now has been synonymous with “Web Services”. I use SOA more precisely: to invoke Web Services using BPM tools and languages. This is an important distinction. SOA is expected to make a significant contribution to the future of Systems Development technologies as indicated in the following paragraphs.

Before SOA, systems development used Workflow Diagrams or Systems Flowcharts that were drawn and then printed, so that relevant business logic could be coded by hand. These manually-coded programs were laboriously tested and eventually deployed for execution. With SOA using BPM tools, this manual coding and testing step is bypassed. Instead, the diagrams are tested for correct logical execution using simulation methods. Once correct, these diagrams are then automatically generated as XML-based BPM languages for immediate execution.

This BPM technology is a major advance in the productivity of systems development; as significant as the development of high-level language compilers in the late 1950's. It becomes easy to invoke Web Services anywhere in the world and to execute them based on Business Rules. When these rules do change, the relevant logic in the diagrams is changed: these diagrams are then automatically regenerated. This promises to transform totally the way we build systems in the future: from slow, error-prone manual coding to an automated discipline. It will enable enterprises to implement changed Business Rules in minutes or hours, rather than in months or years. Enterprises will then be able to change direction rapidly ... to turn on a penny, so to speak.

In the following sections I will discuss the concepts of BPM languages, including: *Business Process Execution Language for Web Services* (called BPEL4WS or just BPEL) from IBM and Microsoft; *Business Process Markup Language* (BPML) from the Business Process Management Institute (BPMI); and *Business Process Specification Schema* (BPSS) from ebXML.

Back to [Contents](#).

---

## Introduction to Service-Oriented and Event-Driven Architectures

*Service-Oriented Architecture (SOA)* is the term that has emerged to describe executable components – such as Web Services – that can be invoked by other programs that act as clients or consumers of those services. As well as the execution of Web Services, these services can also be complete modern – or even legacy – application programs that can be invoked for execution as a “black-box”. A developer does not need to know how the programs work: only the input that they require; the output they provide; and how to invoke them for execution.

The services are loosely coupled to the client program. They can be invoked based on decisions made by business rules. This means that developers can swap one service out and replace it by another service that is designed to achieve the same or enhanced result without having to worry about their inner workings. Today standard parts in a car can be interchanged without having to strip down the whole car and rebuild it. So today with SOA we have similar flexibility, where existing services can be easily replaced by improved services without having to change the internal logic of monolithic application programs as was necessary in the past. Software categories that provide this SOA flexibility are called *Business Process Management (BPM)*, or *Business Process Integration (BPI)* products.

A further term also describes these BPM and BPI execution environments: *Event-Driven Architecture (EDA)*. This is an approach for designing and building applications where business events trigger messages to be sent between independent services that are completely unaware of each other. An event may be the receipt by the enterprise of a Sales Order transaction from a customer for processing. Or it may be a change in a data value that requires a Purchase Order to be placed with a supplier, when the available quantity of a product in the warehouse falls below a minimum balance threshold.

Because the services in an EDA environment are independent, they are decoupled – as distinct from the loosely-coupled services of the SOA-based approach above. An event source sends messages to middleware software, which matches the messages against the subscription criteria or business rules of programs or services that want to be notified of these events. Messages are typically sent using the publish-and-subscribe approach because this enables simultaneous delivery to multiple destinations.

Business Process Management (BPM) is used for workflow modelling and execution by several products for Enterprise Application Integration (EAI). These include Microsoft BizTalk Server and webMethods Business Integrator. IBM, SeeBeyond, TIBCO and Vignette use similar BPM approaches in their EAI products.

Before SOA, most BPM products used proprietary methods to define process logic in Workflow Diagrams. To overcome these product-focused solutions, an open architecture approach has been defined for interoperability. Several XML languages have emerged: Business Process Execution Language for Web Services (BPEL4WS); Business Process Modeling Language (BPML) and the ebXML Business Process Specification Schema (BPSS) definitions.

Back to [Contents](#).

---

## Business Process Execution Language (BPEL)

Recognizing the complexity of accessing web services in synchronous and asynchronous environments, in August 2002 IBM, Microsoft and BEA introduced their jointly defined *Business Process Execution Language for Web Services* (BPEL4WS – or just BPEL). Its specification is now being managed by OASIS. (The BPEL Specifications are at: <http://www-106.ibm.com/developerworks/library/ws-bpel/> and also on OASIS at <http://www.xml.org/>.)

BPEL combines capabilities of IBM's *Web Services Flow Language* (WSFL) from IBM WebSphere Business Integrator and those of Microsoft's XLANG as used by Microsoft BizTalk Server 2002. BPEL includes WSFL support for graph-oriented processes, with XLANG support of structural constructs for processes. BPEL is designed to support implementation of any complex business process, as well as being used to describe interfaces of business processes.

WS-Coordination and WS-Transaction specifications have also been defined for use with BPEL. These offer transaction and process coordination and recovery to address typical error conditions.

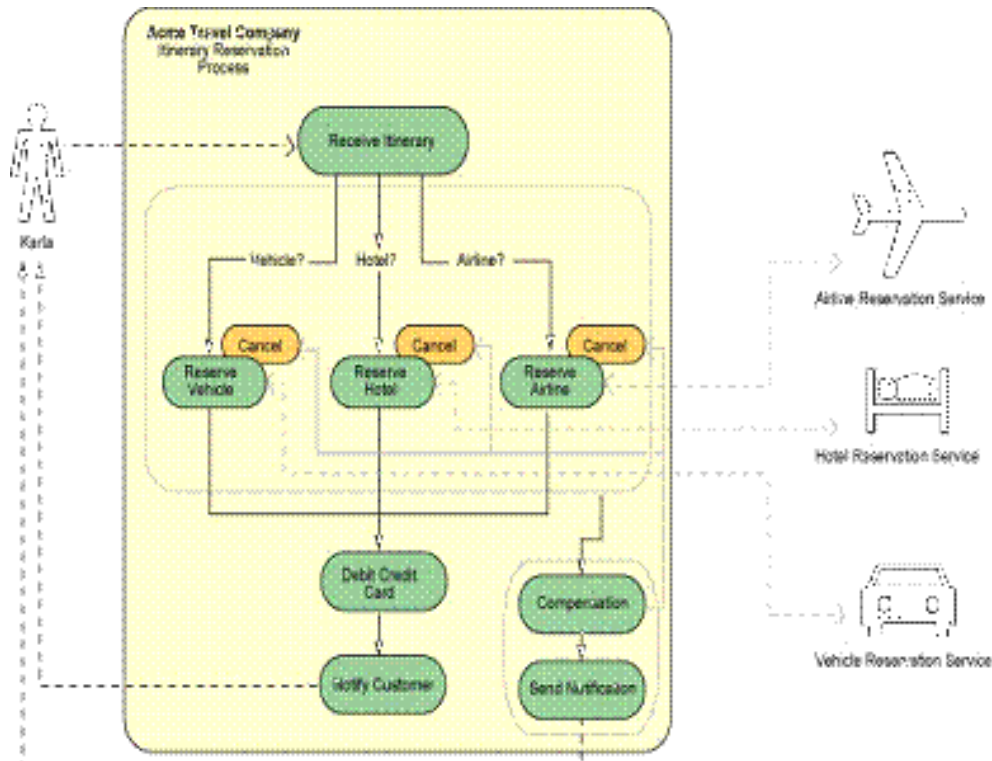
BPEL is a comprehensive workflow definition execution language specified in XML. It can be defined as a programming language and executed directly, but is more likely to be automatically generated from Workflow Diagrams. The BPEL language commands are called *Activities*. Some Activity constructs are discussed next:

- Invoke an operation on a Web Service (<invoke>)
- Wait for an external message (<receive>)
- Generate a response for input / output (<reply>)
- Wait for some time (<wait>)
- Copy data between locations (<assign>)
- Indicate that an error occurred or something went wrong (<throw>)
- Terminate the entire service instance (<terminate>)
- Do nothing (<empty>)
- Define a sequence of steps to be executed in a specific order (<sequence>)
- Branch using a "case-statement" (<switch>)
- Define a loop (<while>)
- Execute one of several alternative paths (<pick>)
- Indicate that steps should be executed in parallel (<flow>)
- Indicate fault logic processing via <throw> and <catch>
- Define *compensation* for error recovery – implement compensating actions for any irreversible actions in error
- Fault handling and compensation can be supported recursively by specifying the relevant scope of execution.

White Papers on BPEL are available from: IBM at <http://www.ibm.com/>; Microsoft at <http://www.microsoft.com/>; and OASIS at <http://www.xml.org/>. The examples in this section have

been drawn from the IBM DeveloperWorks site at <http://www-106.ibm.com/developerworks/library/ws-bpelwp/> and also at <http://www-106.ibm.com/developerworks/library/ws-bpelcol1/?n-ws-8292>.

BPEL is a comprehensive workflow definition execution language that is specified in XML. It can be written as a programming language and executed directly. But it is mainly intended for automatic generation and execution directly from Workflow Diagrams. The IBM DeveloperWorks web site has a White Paper that describes two examples: a Loan Processing example at <http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol2/> and a Travel Itinerary example that is shown in Figure 1.



Source: <http://www-106.ibm.com/developerworks/library/ws-autobp/>

**Figure 1: A Travel Example using BPEL (Source: See URL above)**

Figure 1 shows the integration of reservation Web Services from airline, hotel and car rental partners in a Travel Itinerary business process. The White Paper referenced in Figure 1 details how the activities of a business process are externalized as Web services, such as the initial wait for receipt of a customer itinerary request, for example. The coordination activities for multiple Web services within a business transaction are described, together with the dynamic linking to services from multiple providers at runtime. This is based on data that is derived from the process flow itself: for example, which airline the customer wishes to use, the preferred car rental company, and a requested hotel.

The BPEL, WS-Coordination and WS-Transaction Specifications from the OASIS web site at <http://www.xml.org/> describe how they are used together. *WS-Coordination (WS-C)* is an extensible framework for coordinating the actions of distributed applications operating in a heterogeneous environment. WS-C is used to create an environment to propagate an

activity to other services and so coordinate their actions, or to register for coordination protocols. It defines coordination types that specify a set of coordination behaviors.

*WS-Transaction (WS-T)* specifies two coordination types used in conjunction with *WS-Coordination: Atomic Transaction (AT)*; and *Business Activity (BA)*. The AT behavior is used to coordinate activities that have a short duration, while BA behavior coordinates activities that are long in duration and so need to apply business logic to handle business exceptions.

With development of BPEL, WS-C and WS-T now managed by the OASIS WSBPEL Technical Committee, its members include Booz Allen Hamilton, BEA, CommerceOne, E2open, EDS, IBM, Microsoft, NEC, Novell, Oracle, SAP, SeeBeyond, Siebel, Sun, Sybase, TIBCO, Vignette, Waveset and others. With the strength of these organizations behind its adoption, BPEL is expected to become a major force in XML-based languages for Business Process Management.

Back to [Contents](#).

---

## Business Process Modeling Language (BPML)

Business Process Modeling Language (BPML) is complementary to Business Process Execution Language (BPEL). BPML can be used to define the detailed business process behind each service. Details about the Business Process Modeling Language (BPML) specifications are available from the BPMI web site at <http://www.bpmi.org/>. BPML maps business activities to message exchanges. It can be used for the definition of enterprise business processes, the definition of complex Web Services, and the definition of multi-party collaborations. Some of the organizations that are involved in the definition of the BPML Specification are CSC, Intalio, SAP, Sun, SeeBeyond and Versata. From the BPML Specifications, its intent is as follows:

*“Business Process Modeling Language (BPML) is an XML language to define a formal model for expressing executable processes that address all aspects of enterprise business processes. BPML defines activities of varying complexity, transactions and compensation, data management, concurrency, exception handling and operational semantics. BPML provides a grammar in the form of an XML Schema to enable the persistence and interchange of definitions across heterogeneous systems and modeling tools.”*

BPML is a rich and mature language to express both simple as well as complex business processes. BPML and BPEL share an identical set of idioms and similar syntaxes as block-structured languages. Compared to the activities supported by BPEL as listed last month, BPML syntax supports: Activities and Activity Types; Processes; Properties; Signals; Schedules; and Exceptions. To illustrate, Simple and Complex BPML Activity Types are listed and discussed next.

## Simple BPML Activity Types

- *action*: Performs or invokes an operation involving the exchange of input and output messages.
- *assign*: Assigns a new value to a property.
- *call*: Instantiates a process and waits for it to complete.
- *compensate*: Invokes compensation for the named processes.
- *delay*: Expresses the passage of time.
- *empty*: Does nothing.
- *fault*: Throws a fault in the current context.
- *raise*: Raises a signal.
- *spawn*: Instantiates a process without waiting for it to complete.
- *synch*: Synchronizes on a signal.

## Complex BPML Activity Types

- *all*: Executes activities in parallel.
- *choice*: Executes activities from one of multiple sets, selected in response to an event.
- *foreach*: Executes activities once for each item in an item list.
- *sequence*: Executes activities in sequential order.
- *switch*: Executes activities from one of multiple sets, selected based on the truth value of a condition.
- *until*: Executes activities once or more based on the truth value of a condition.
- *while*: Executes activities zero or more times based on the truth value of a condition.

A *Complex Activity* is an activity that contains one or more child activities. It establishes a context for execution and directs that execution. Complex activities define hierarchical composition. This can be as simple as repetitive execution of a single activity, or a means to establish a nested context for the execution of multiple activities. BPML supports other forms of composition, which include cyclic graphs and recursive composition. Complex activities are used when hierarchical composition is required, in particular to establish a new context for the execution of child activities.

A *Simple Activity* is any activity that may lead to the execution of multiple activities, specifically the *action*, *call*, *compensate* and *spawn* activities. However, a simple activity does not by itself define the context for the execution of other activities. Differentiating between these further, the following language summary illustrates that BPML includes all of the logic constructs of a rigorous programming language.

A complex activity that contains multiple activity sets must of course select which one to use. Several typical logic constructs are used. The *choice* activity waits for an event to be triggered and then selects the activity set associated with that event handler. The *switch* activity evaluates conditions and selects the activity set associated with a condition that evaluates to true. All other complex activities defined in the BPML specification contain a single activity set and so do not have to make such decisions.

A complex activity also determines the number of times to execute activities from the total activity set. Typical logic constructs here are: the *until* activity, which repeats executing activities until a condition evaluates to true; the *while* activity, which executes activities repeatedly while the condition evaluates to *true*; and the *foreach* activity. This repeats executing activities, once for each item in the item list. All other complex activities above execute activities from the activity set exactly once.

A complex activity determines the order in which activities are executed. The *sequence* activity executes all activities from the activity set's list in sequential order. The *all* activity executes all activities from the activity set's list in parallel. All other complex activities defined in BPML execute activities in sequential order.

The complex activity completes when it has finished executing all activities from the activity set. This includes all activities that are defined in the activity list and all processes instantiated from a definition made in the activity set's context. Nested processes and exception processes are considered activities of the activity set.

Simple activities *abort* and throw a *fault* if they cannot complete due to unexpected error. Complex activities *abort* and throw a *fault* if one of their activities throws a fault from which they cannot recover.

With its additional support for nested processes and other syntax, BPML is a superset of BPEL. When BPEL is used with BPML, an end-to-end view depicts the role of each individual business process in the overall choreography, and the business activities that are performed by each role.

BPEL and BPML are similar approaches to solve the same problem: the definition of process logic in XML so that it can be used as executable code by BPM-based software products. These languages are evolving. At the time of writing, BPML had reached the level of a W3C Proposed Recommendation. A decision between them has not yet been resolved. All are solutions to the same problem, with specifications and languages that are conceptually similar; they may in time consolidate into one overall specification.

The cross-participation of organizations in two (and for some organizations all three) of these initiatives augurs well for possible convergence of the specifications. But until that time, most BPM tools will most likely need to support generation of all three BPM languages from workflow models or process diagrams, including BPSS as discussed next.

Back to [Contents](#).

---

## ebXML Business Process Specification Schema (BPSS)

This is the third BPM language that we will discuss. BPSS is part of ebXML, which has decades of experience in *Electronic Data Interchange* (EDI) behind its Specification. The full ebXML Specifications on <http://www.ebxml.org/> and also on OASIS at <http://www.xml.org/> should be read in conjunction with the discussion of ebXML BPSS Specification in this

section.

BPSS differs from BPEL and BPML. It defines a business process for physical business interchanges between parties for collaboration and for transactions to be carried out between commercial business partners. It is designed to work in conjunction with the *ebXML Collaboration Protocol Profile (CPP)* and *Collaboration Protocol Agreement (CPA)*, both defined in the ebXML Specifications. In contrast, the other BPM languages are generic to Web Services: they do not provide explicit commercial semantics (or terms) as discussed next, nor do they have the intent of a trading partner agreement as with ebXML.

A Web Service may have to be designed for internal and external uses. For this reason, internal-use Web Services can be described in the other BPM languages, while external-use Web Services may require further specification: described with commercial terms for business collaboration using ebXML BPSS.

For example, if a Supplier accepts a Purchase Order from a Customer, this acceptance is a binding legal agreement: for the Supplier to deliver the requested Products or Services to the Customer at an agreed price and time; and for the Customer to pay the Supplier following delivery. The various messages and interactions sent between these external businesses constitute a binding contract between both parties. BPSS therefore includes the concepts of time periods for business response, plus *non-repudiation*: neither party can deny its legal obligations to the other party, once the Purchase Order has been issued and accepted.

A company may want to wrap its implementation of a given role in external-use BPSS collaboration, and additionally also as an internal-use Web Service. To achieve this it can describe the Web Service in any BPM language: there is nothing in the specifications that prevent either of these two scenarios; but neither BPSS nor the other BPM specifications have been designed with the other approaches in mind.

BPSS also specifies how models are used to generate XML-based BPSS definitions. It describes the XML-generation concepts using UML (Unified Modeling Language) class diagrams and action diagrams. With its earlier start from 1999, the specification of ebXML BPSS is several years ahead of the other BPM languages.

The ebXML Business Process Specification Schema provides a standard framework for business process specification. It works with the Collaboration Protocol Profile (CPP) and Collaboration Protocol Agreement (CPA) of the *ebXML Specifications* to bridge the gap between business process modeling and ebXML compliant e-commerce software, such as *ebXML Business Service Interface (BSI)* software.

The architecture of the ebXML Business Process Specification Schema consists of the following functional components:

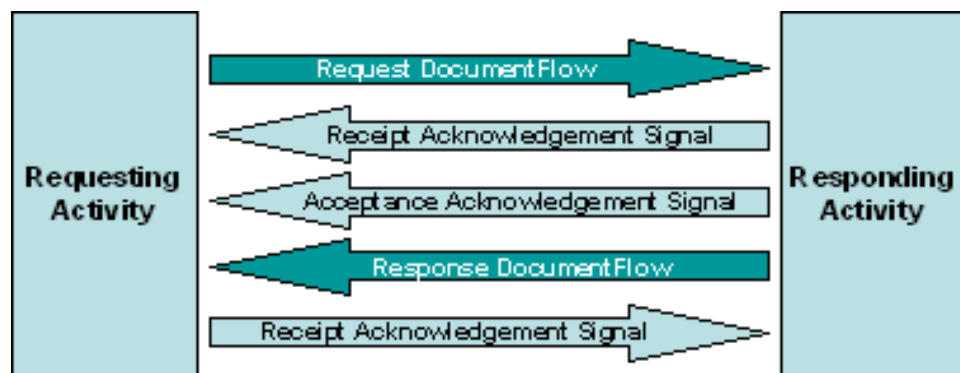
- UML version of the Business Process Specification Schema;
- XML version of the Business Process Specification Schema;

- Production Rules defining the mapping from the UML version of the Business Process Specification Schema to the XML version;
- Business Signal Definitions (that acknowledge receipt of messages).

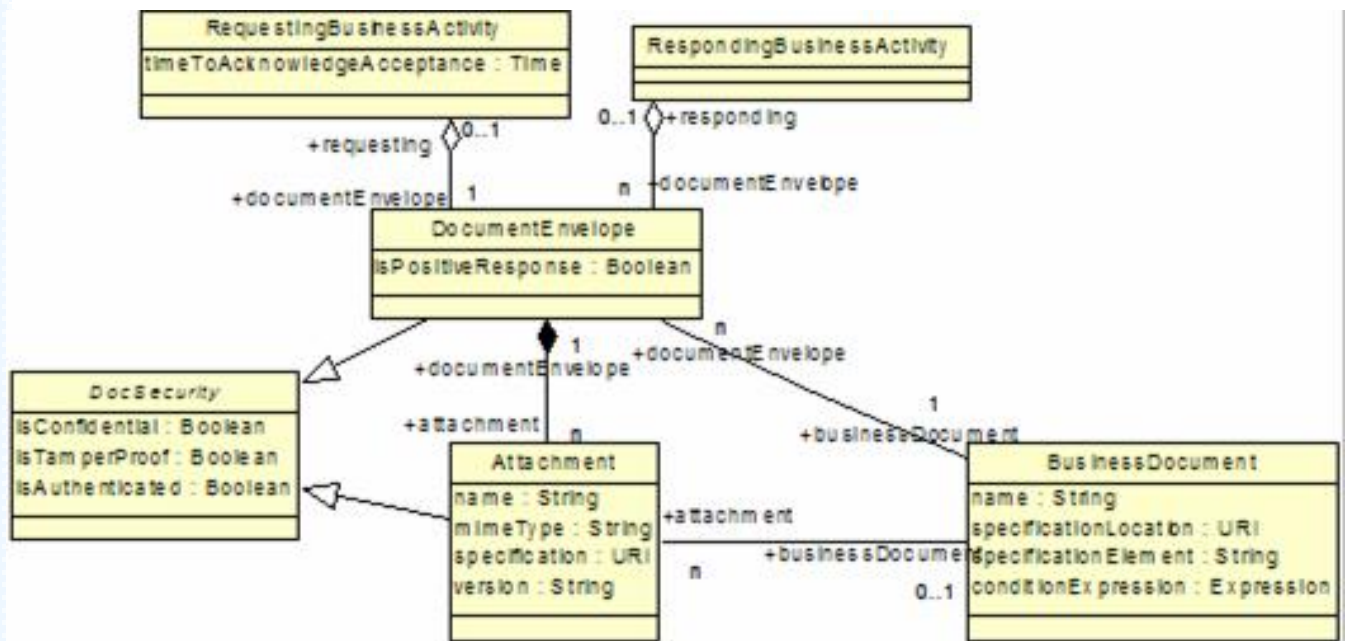
Together these components allow complete specification of the run-time aspects of a business process model.

Business Process Specification Schema is the machine-interpretable run-time business process specification that is needed by ebXML *Business Service Interface* software. The BPSS code is incorporated with or referenced by the ebXML trading partner CPP and CPA. Each CPP declares its support for one or more Roles within the Business Process Specification, documented by UML Use Case diagrams. Within these CPP profiles and CPA agreements are added further technical parameters that result in a full specification of the run-time by *ebXML Business Service Interface* software at each trading partner.

To illustrate the generation of BPSS, Figure 2 shows business document and receipt document flows that are acknowledged Business Signals. The request requires business signal acknowledgement of the receipt and acceptance, while the response requires receipt acknowledgement.



**Figure 2: Document Flows and Signals, with Sequence (Source: ebXML BPSS Specifications)**



**Figure 3: UML Class Diagram for Document Flow in Figure 2 (Source: ebXML BPSS Specifications)**

Figure 3 shows a UML Class Diagram for the document flow in Figure 2. Using this, the BPSS XML code in Figure 4 has been generated automatically from the Class Diagram in Figure 3. The BPSS Specification provides full guidance for automatic generation by UML modeling tools of XML-based BPSS code from class diagrams and action diagrams.

```

<BusinessTransaction name="Create Order">
  <RequestingBusinessActivity name=""
    isNonRepudiationRequired="true"
    timeToAcknowledgeReceipt="P2D"
    timeToAcknowledgeAcceptance="P3D">
    <DocumentEnvelope
      BusinessDocument="Purchase Order"/>
  </RequestingBusinessActivity>
  <RespondingBusinessActivity name=""
    isNonRepudiationRequired="true"
    timeToAcknowledgeReceipt="P5D">
    <DocumentEnvelope isPositiveResponse="true"
      BusinessDocument="PO Acknowledgement"/>
  </RespondingBusinessActivity>
</BusinessTransaction>

```

**Figure 4: Partial BPSS XML Syntax for Document Flows in Figure 2 (Source: ebXML BPSS Specifications)**

Figure 4 shows the BPSS definition for a *Create Order* business transaction. To define the commercial and legal nature of this transaction, the *RequestBusinessActivity* attribute shows: *isNonRepudiationRequired="true"* with the *timeToAcknowledgeReceipt="P2D"* and

*timeToAcknowledgeAcceptance="P3D"* (where "P2D" is a W3C Schema syntax standard that means Period=2 Days; P3D means Period=3 Days). These periods are all measured from the original sending of the request.

In time it is expected that the other BPM languages – or a consolidated version of them – will evolve along similar lines to BPSS, but with broad applicability to all applications: not just for EDI and ebXML-based applications. In fact BPSS is a clear pointer to a future when application development will be based on automatic generation of XML-based languages directly from workflow or other process models.

Back to [Contents](#). .

---

## AUTHOR

Clive Finkelstein is the "Father" of Information Engineering (IE), developed by him from 1976. He is an International Consultant and Instructor, and Managing Director of Information Engineering Services Pty Ltd (IES) in Australia.

Clive Finkelstein's books, online interviews, courses and details are available at <http://www.ies.aust.com/cbfindex.htm>.

For More Information, Contact:

Clive Finkelstein, Managing Director  
Information Engineering Services Pty Ltd  
PO Box 246, Hillarys, Perth WA 6923 Australia

Details: <http://www.ies.aust.com/cbfindex.htm>

Web Site: <http://www.ies.aust.com/>

Phone: +61-8-9402-8300

Fax: +61-8-9402-8322

Email: [cfink@ies.aust.com](mailto:cfink@ies.aust.com)

---

| [Home](#) | [Courses](#) | [Certification](#) | [Projects](#) | [Papers](#) | [TEN Archive](#) |  
| [Online Store](#) | [Contact Us](#) | [Search](#) |

(c) Copyright 2004-2006 Information Engineering Services Pty Ltd. All Rights Reserved.