

An Approach to Moving Industry Business Messaging Standards to Web Services

(CIDX, OAGi, PIDX & RosettaNet)

Disclaimer:

This document is distributed without any warranty. The information in this document pertaining to existing standards and practices are correct to the best of our knowledge, but IBM assumes no responsibility for the accuracy of the information provided here and use of such information is at the recipient's own risk. IBM makes no warranties or conditions either express or implied, including but not limited to, the implied warranties or conditions of merchantability, fitness for a particular purpose, and non-infringement, regarding the document or technical support, if any. Under no circumstances is IBM liable for any of the following, even if informed of their possibility: loss of, or damage to, data; special, incidental, or indirect damages, or for any economic consequential damages; or lost profits, business, revenue, goodwill, or anticipated savings. IBM provides no assurances that any reported problems may be resolved with the use of any information that IBM provides. By furnishing information, IBM does not grant any licenses to any copyrights, patents or any other intellectual property rights. IBM may also make improvements to the information provided here at any time without notice.

An IBM White Paper on Using Web Services within Industry

IBM Authors and Contributors (alphabetically):

Paul Bunter
Ralph Hertlein
Sreedhar Janaswamy
Rania Khalaf
Keeranoor Kumar
Michael McIntosh
Anthony Nadalin
Shishir Saxena
Peter Williams

An Approach to Moving Industry Business Messaging Standards to Web Services Version 1.0

Table of Contents

1	Abstract.....	3
2	Industry Business Messaging Standards Today	4
3	The Benefit of Web Services	6
4	The Web Services Family of Standards	8
5	A Web Services Approach to Industry Messaging.....	10
	5.1 Define Business Messages Using XML Schema	11
	5.2 Define Web Services Messages and Operations Using WSDL	12
	5.3 Define Web Services Message Handling Using BPEL	14
	5.4 Define Web Services Quality of Service Using Policy Files	16
	5.5 The Web Services Runtime Environment	17
6	Next Steps	20
7	References.....	22
Appendix A	The BPEL Solution In Depth	24
	A.1 Basic BPEL	24
	A.2 Managing Message Instances	26
	A.3 Timeouts	27
	A.4 Retries	28
	A.5 Validating Messages	28
	A.6 Faults.....	29
	A.7 Duplicate, Out-of-Order, and Post-Completion Messages	29
	A.8 Default Executable BPEL Processes.....	31

1 Abstract

This paper addresses aspects of adapting various Industry Standards-based Business Messaging Systems to leverage the benefits of Web Services. It reviews the current functionality of Industry Standards such as CIDX, OAGi, PIDX and RosettaNet that use the RosettaNet Implementation Framework (RNIF), and the benefit of adapting them to a Web Services environment. With this foundation established, it proposes an approach that maintains the existing functionality using Web Services tools, methods and standards available today or in the near future. This paper does not presume that this is the only solution, nor does it presume that there will be no further improvements with Web Services. It offers a recommendation that only matches those standards and technologies available in the near term that can be used to develop a solution. A future paper will explore the deeper benefits that can be gained by fully leveraging Web Services.

The approach outlined here applies to both the business messages and documents that contain business content, and to the RNIF which provides the messaging infrastructure needed to exchange the business messages. For business content, it is proposed that each business message package consist of a set of XML Schemas, WSDL definitions, WS-Policy assertions, and abstract BPEL processes. The processes and infrastructure embodied within the RNIF are also proposed to be handled by the abstract BPEL processes along with other elements of a Web Services stack, including WS-Security, WS-ReliableMessaging, WS-Addressing, WS-Policy and WS-PolicyAttachment. The solution aims to comply with WS-I profiles and specifications where possible to maximize interoperability.

2 Industry Business Messaging Standards Today

The majority of the Industry Business Messaging Standards in use today are developed by consortia of companies primarily from within specific industries. Their goal is to develop open standards and processes that enable trading partners to exchange standardized business messages within an e-business environment. Some of the standards such as RosettaNet are supported by major application vendors such as SAP, I2, Manugistics, and Oracle and all major middleware vendors, including IBM. Others are supported by the majority of middleware vendors and industry specific implementations.

Each industry consortium has created a large set of specifications that describe particular business interactions between trading partners. For example, these specifications are known within RosettaNet as *Partner Interface Process* (PIP) specifications and within the OAGIS Standard as *Business Object Documents* (BODs). They typically include information on the sequence of activities, the different roles involved in the interaction (ie, buyer and seller), the structure and content of the business messages to be exchanged, and various quality of service requirements such as timing constraints, security, and non-repudiation.

Currently, RosettaNet PIPs are packaged as a set of XML DTD, HTML, and Microsoft® Word files. The PIP-specific business message formats are defined in the XML DTD files. The specification is given in the Microsoft® Word document and describes message exchange and business control parameters via UML diagrams, tables and text. Note that RosettaNet is in the process of converting its PIPs to use XML Schema. CIDX, OAGi (the organization responsible for OAGIS) and PIDX already package their messages as XML Schemas with XML documentation. A typical list of business messages for any of these consortia includes a minimum subset of *Quote Request*, *Quote*, *Quote Notification*, *Order*, *Order Response*, *Order Change*, *Order Status Request*, *Receipt*, *Advanced Shipped Notice (ASN)*, *Return Requisition Request*, *Field Ticket*, *Field Ticket Response*, *Invoice*, *Invoice Response*, and *Invoice Exception*. OAGi and RosettaNet have much larger sets of messages covering, in addition, areas such as supply chain forecasting, product data exchange, manufacturing, design-win and product marketing.

The Business Message is the construct that is exchanged between trading partners and consists of multiple message components packaged together in a MIME *multipart/related* document. Typically, this consists of header documents, the business action / signal message (Service Content), and optional attachments. The RNIF, used by multiple consortia for their transport, defines the structure of the Business Message and the steps required for transmitting the message between trading partners. This includes message packaging and unpackaging, transmission protocols, error handling, and validation of some content.

An Approach to Moving Industry Business Messaging Standards to Web Services Version 1.0

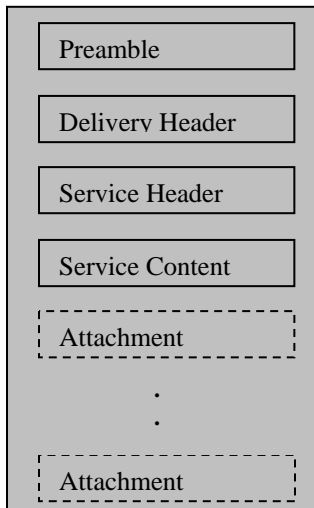


Figure 2-1: Example of current structure of a typical RosettaNet Business Message

Now with several years of experience, the industry consortia are able to reflect on theory versus practice and also observe new business trends. For example, RosettaNet specification packaging makes installation more difficult than is desired. This impacts major users from scaling their e-business networks as rapidly as they would like. Selecting relevant information from the complex specification text is labor intensive and error-prone. With RosettaNet beginning to move to XML Schema specifications, this paper's proposals are timely, as schema based PIPs will be able to leverage Web Services as easily as the other consortia's messages. For the other consortia that are already schema based, it can be used to guide their transition from RNIF to Web Services.

At a minimum, Web Services offers an alternate channel to RNIF for exchanging business objects or messages. This alone makes it a logical candidate for consideration by these industry consortia. However, other compelling reasons exist for them to offer a Web Services option.

3 The Benefit of Web Services

In the global landscape of business, enterprises conducting e-business among dynamic supply chains must be capable of equally dynamic communication from end-to-end, both upstream and downstream. Internal applications must be able to communicate information quickly and easily both intra and inter-enterprise. This requires a higher level of automated application integration.

Inter-enterprise business communications within increasingly disaggregated value chains effectively mandate the use of open standards. Open standards permit accurate, effective communication to be established among all partners within the product life cycle that transcend enterprises, vendors, infrastructures and platforms. Today, the market is choosing XML as the framework to enable heterogeneous messaging. However, there are still many more components needed to fulfill the range of capabilities business partners require to successfully automate the value chain.

Web Services hold the promise to answer the market demand for those additional components. Although the definition of Web Services sparks much discussion, there appears to be general agreement around the following: *Web Services are modular applications that can be described, published, located, assembled, invoked and controlled using XML over a network.* Using this definition, Web Services can be viewed as business service building blocks that can be assembled across multiple heterogeneous networks to create desired functionality. By providing remote application data and functions where desired, as if they were local, web services can offer the following benefits:

- They enable the creation and packaging of discrete *service modules* which can be linked into more complete business processes. Various linked modules in multiple combinations permit trading partners to answer their own questions and needs, in their own timeframe, by leveraging their partners' systems as if they were applications resident within their own system. Partners can then *extend* that information up or down chain quickly and easily with their own Web Services, offering the ability to extend the enterprise much faster and cheaper than the methods in use today. This in turn offers companies the ability to participate in specific supply chains faster and more easily; and as more enterprises field web services, to move across supply chains, greatly expanding the opportunities for any given enterprise.
- They provide the ability to define and control functions at a more granular business process level and then aggregate those granular functions into more encompassing transactions. For example, rather than performing separate operations to query multiple catalogs, request multiple quotes and submit a purchase order, one could aggregate these transactions to create a single operation to query a set of manufacturers and then buy on specification, price, and

An Approach to Moving Industry Business Messaging Standards to Web Services Version 1.0

availability. This capability offers business and standards the opportunity to converge to a more service focused methodology.

- Web Services also offers tools and methods that can substantially automate solution development – reducing the costs and time needed to implement – thus expanding the range of companies that can be directly connected into high speed markets.

When Web Services are combined with industry consortia business level semantics, the value becomes very evident. Together Web Services and these industry standards can simplify the connection and communications of multiple business tiers, even down to unique applications. Such a combination will not only offer enterprise adaptability, it will also enable significant process changes as information is more quickly shared across a supply chain – increasing responsiveness of and visibility into more partners. Web Services and industry standards will enable supply chains to achieve higher levels of automation and responsiveness without having to invest in proprietary infrastructures. Together they can offer a direction to the ever demanding marketplace.

4 The Web Services Family of Standards

The Web Services framework consists of a modular, extensible stack of XML specifications and standards targeting the emerging infrastructure in which distributed, heterogeneous applications are exposed by different organizations as services on the Internet. These services have their capabilities described and published in a machine readable format.

Two key concepts differentiate Web Services from other approaches: the composability and extensibility of the specifications making up the stack, and the interoperability and portability obtained through standardization efforts. Web Services, by design, do not dictate a particular implementation or system architecture, thereby enabling their use across systems that support the XML standards and specifications making up the framework. They offer a much more automated approach to setup and use through standards compliant tooling. Any party with a compliant implementation can use them to carry out business interactions.

The approach described in this paper will use a subset of the Web Services specifications that enable services to be described, interacted with, and composed: WSDL, WS-BPEL, WS-Security, WS-ReliableMessaging, WS-Addressing, WS-Policy, WS-PolicyAttachment, and WS-SecurityPolicy.

The Web Services Description Language (WSDL) provides a standard way to describe the interface supported by a Web Service, the binding of this interface to an interaction protocol, and addressing information on where an implementation is deployed. Each WSDL definition consists of an abstract part and a concrete part. The abstract part of WSDL consists of a set of typed messages and a set of portTypes. A *portType* is simply a named logical grouping of operations. Each operation has a name and defines the input and possible output WSDL message(s) it expects for an interaction. The concrete parts of a WSDL definition consist of bindings that bind portTypes to specific interaction protocols and message formats (ie, HTTP and SOAP). Finally, a port provides the endpoint information (ie, URL) for accessing an implementation of a portType using a certain binding (and can be addressed using WS-Addressing). A set of ports may be grouped as a service.

The Business Process Execution Language for Web Services (BPEL4WS aka WS-BPEL, or simply BPEL) is a workflow language for composing Web Services by providing control semantics around a set of interactions with the services being composed. The business process itself may also be exposed as a Web Service, providing its own WSDL portTypes to its partners.

BPEL processes come in two flavors: *executable* and *abstract*. The executable variant contains all the information necessary for a process to be executed by a workflow system. On the other hand, the abstract variant enables one to define interaction protocols or

An Approach to Moving Industry Business Messaging Standards to Web Services Version 1.0

process templates that partners can use to know how to interact with the process and how it will interact with them. Certain information that is considered either private or ungoverned by the business logic can be left out of an abstract process. This includes such items as the initialization of variables or calling private routines.

BPEL processes can invoke and be invoked by one another since a process is just another Web Service with a WSDL definition. BPEL's recursive composition and its use of the same language for defining both executable and abstract processes permit the standardization of business processes at the abstract level, while enabling all the adaptability needed for partners to define their own compliant executables that can call into their private functionality. BPEL also provides fault, event and compensation handling capabilities.

Protocols for carrying out quality of service requirements have been defined for a number of areas related to Web Services. Of these, we highlight security and reliable messaging. WS-Security provides for confidentiality, authorization, and integrity of messages using digital signatures and encryption. WS-ReliableMessaging provides a protocol for reliable message delivery.

In Web Services, quality of service parameters may be defined externally as assertions using WS-Policy and attached to different parts of a WSDL definition using the WS-PolicyAttachment specification. WS-Policy assertions related to WS-Security are defined in the WS-SecurityPolicy specification. WS-Policy assertions related to WS-ReliableMessaging are defined within the WS-ReliableMessaging specification itself.

Using the above set of standards and specifications, we now propose an approach for defining and exchanging Business Messages using Web Services.

5 A Web Services Approach to Industry Messaging

Our proposed Web Services approach consists of two main parts: modernization and repackaging of Business Messages, and the use of a Web Services run-time stack to support message exchange and handling.

Modernization and repackaging of the Business Messages includes conversion of document definitions from XML DTD to XML Schema for all header documents (e.g., *Preamble*, etc.) and business signal documents defined in RNIF (e.g., *Receipt Acknowledgment*, etc.) as well as all business action documents defined by each message specification (e.g., the documents defined in PIP 3A4, etc.) This effort will also address the harmonization of such schema constructs as namespaces and business element reuse. Efforts such as these are already underway at various levels within many standards bodies and are being harmonized and funneled through organizations such as WS-I. Then, for each role defined in the Business Message specification (for example, buyer and seller), a set of WSDL definitions, abstract BPEL processes, and WS-Policy assertions are generated to define the structure of the messages sent and received via Web Services, as well as the handling of those messages for the role. As in the current PIP packaging, a human-readable document should also be included.

Each interacting party in a business interchange would go to a repository and pick up the set of files related to the role it will play in the interaction. This is similar to the downloading of a specification today except the recommended downloads will be fully machine processable by available tooling. Using such tooling, they may then modify or customize the BPEL and other files based on local policy, Partner Profiles and Trading Partner Agreements before deploying them. These new recommended specifications will begin to resolve issues currently being addressed by efforts such as the RosettaNet Trading Partner Profile work.

The corresponding runtime environment consists of a WS-BPEL sub-system to run implementations of the abstract processes, a WS-Security sub-system, a WS-ReliableMessaging sub-system, a non-repudiation sub-system, and a message validation sub-system. The first three sub-systems are part of the Web Services stack and are covered by published specifications. The non-repudiation and message validation sub-systems represent opportunities for standards consortia to address their standardization, or they can be implemented as internal / private processes.

Note that this proposal is expecting support for non-repudiation to be contained within WS-Security. We propose a profile that could extend Web Services Security to provide *Proof of Message Origin* and *Proof of Message Receipt*. This solution would serve the non-repudiation needs of business messaging and handle them appropriately in the middleware layer of a WS-* compliant system.

5.1 Define Business Messages Using XML Schema

In creating a Web Services-based business message, one must first create the relevant XML Schema definitions. They may either be converted from existing DTDs or directly defined, as some of the consortia have done for ones that are natively Web Services-enabled. This effort will need to be rationalized with those efforts being brought forth by other standards organizations such as the WS-I.

RosettaNet will have an additional challenge in that existing PIP specifications define additional restrictions on the use and content of schema elements. Existing schema technology and tools can, add the first level of restrictions directly to the XML Schema definitions. Additional restrictions can then be applied via tooling during subsequent layers of processing. This newer approach, although counter-intuitive for a standard, has already been applied by other standards bodies. They leverage the tooling needed for meeting partner requirements in later stages of processing, rather than imposing a construct within the standard. Note that some of the complex business requirements, beyond those covered within schema, can be handled by internal sub-systems and applications, as well as addressed by the standard through combined efforts with other standards organizations.

In addition to the action messages defined in the Business Message specification, XML Schema definitions must also be generated for the standard header messages (*Preamble*, *Delivery Header*, and *Service Header*) and business signal messages (*Receipt Acknowledgment* and *Exception*) that are currently defined in RNIF.

```
<xsd:element name="Pip3A4PurchaseOrderRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:fromRole"/>
      <xsd:element maxOccurs="1" minOccurs="0"
        ref="tns:GlobalDocumentFunctionCode"/>
      <xsd:element ref="tns:PurchaseOrder"/>
      <xsd:element ref="tns:thisDocumentGenerationDateTime"/>
      <xsd:element ref="tns:thisDocumentIdentifier"/>
      <xsd:element ref="tns:toRole"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 5-1: Part of a sample XML Schema for the 3A4 Purchase Order Request action message

Each Business Message will be formed from a set of XML Schema types, either complex or primitive. This corresponds, in Web Services, to defining a WSDL message that has

parts with the matching Schema types. An important decision in the creation of the XML Schema types, therefore, would be reusability. This modularity allows that a Schema type may be used in multiple places in one WSDL message, multiple WSDL messages, and even WSDL messages that are logically tied to multiple business operations. The level of granularity, at which these Schema types can be created, is therefore an area with some flexibility and choice for each consortium. The optimal choice will depend on a study of business needs across the business process, the similarities between them, and the capabilities of current or near term tooling.

With respect to reusability and interoperability, there is some debate regarding optimal use of XML namespaces. This issue is being prototyped in several standards bodies throughout the market and in WS-I working groups. Participation in these debates and groups will be a requirement for any industry consortium to pace industry directions.

5.2 Define Web Services Messages and Operations Using WSDL

The Business Message package will typically contain one WSDL document for each role defined in the interaction. Each WSDL document only includes the abstract section of a WSDL definition, namely the messages, operations, and portTypes. The portType will contain the operations that the given party will expose to the other party. A message exchange for a typical standard message would therefore correspond to the invocation of a Web Services operation from such a portType.

In the case of synchronous actions, one party would provide an input-output WSDL operation that takes the requesting message as input and returns the response message as its output. For asynchronous actions, a message exchange from Trading Partner A to Trading Partner B corresponds to B providing an input-only WSDL operation that can consume that message. If there is a requirement for an acknowledgement to be returned, then, in a similar manner, A would provide an operation that can consume this acknowledgement.

For example, the RosettaNet PIP 3A4 *Request Purchase Order* incorporates a buyer role and a seller role within its envisioned process. In order for such a buyer to send a purchase order to the seller, the seller must expose a *receivePurchaseOrder* operation on its portType that consumes a WSDL message containing the PurchaseOrder. Other consortiums such as OAGIS have included operations such as *Get* and *Show* within their specifications and they would also need to mirror this methodology for executing asynchronous processes.

An Approach to Moving Industry Business Messaging Standards to Web Services Version 1.0

```
<message name="PurchaseOrderRequestMessage">
  <part name="Preamble" type="preamble:Preamble"></part>
  <part name="DeliveryHeader" type="delivery:DeliveryHeader"></part>
  <part name="ServiceHeader" type="service:ServiceHeader"></part>
  <part name="Pip3A4PurchaseOrderRequest" type="pip3a4req:Pip3A4PurchaseOrderRequest"></part>
</message>

<portType name="PIP3A4ReqServicePort">
  <operation name="PurchaseOrderRequest">
    <input message="tns:PurchaseOrderRequestMessage"></input>
  </operation>
  <operation name="Exception">
    <input message="tns:ExceptionSignal"></input>
  </operation>
  <operation name="ReceiptAcknowledgment">
    <input message="tns:ReceiptAckSignal"></input>
  </operation>
</portType>
```

Figure 5-2: Part of a sample WSDL definition for the 3A4 Purchase Order Request action message

The binding and port information, which specify how and where to call the operations, will not be added until the WSDL definitions are deployed by the given party. Although bindings and ports are out of the scope of the current Business Message specifications, we recommend requiring at least a WS-I compliant SOAP binding.

A SOAP representation of the Business Message would be constructed using the WSDL message definitions. The SOAP headers should contain composable elements for WS-Security, WS-ReliableMessaging, and WS-Addressing to address the functions currently in use with the RNIF today. The SOAP body will contain all elements of the standard Business Message. Keeping all components of the Business Message together allows easier conversion to / from the RNIF 2.0 format by a trading partner or trusted hub and may provide advantages for message processing and archiving. The SOAP body (along with any attachments) will be signed and may be encrypted using WS-Security. This will provide for message origin authentication, integrity, confidentiality, non-repudiation and message freshness (via *Timestamp*) of the Business Message.

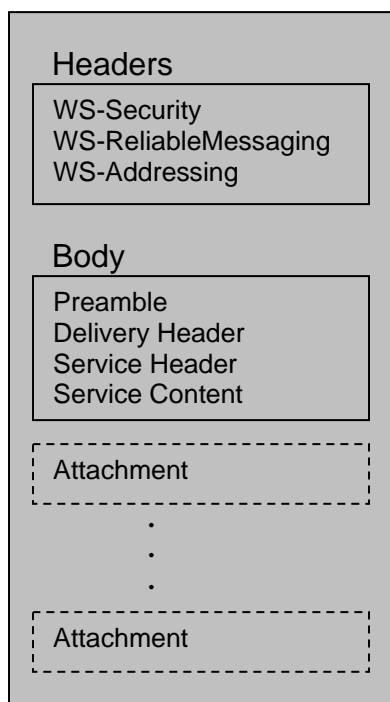


Figure 5-3: Example SOAP message structure for a Business Message

It is proposed that all the resulting specifications strive for maximum interoperability by conforming to the profiles and specifications being offered through the WS-I. At a minimum, compliance with the *WS-I Basic Profile 1.1 – Final Specification*, *Simple SOAP Binding Profile 1.0 – Final Specification* and *Attachments Profile 1.0 – Final Specification* be part of the design of any specification or solution. It is also expected that the specifications from the Basic Security Profile Working Group Draft and the Profile Conformance Framework WG Draft be incorporated, based on the appropriate timing, along with any additional support for interoperability that may emerge from the WS-I.

5.3 Define Web Services Message Handling Using BPEL

A typical specification identifies not only the messages that will be exchanged by the interacting parties, it also lists or infers the sequence in which those messages are exchanged along with any timing constraints. Since the de facto specification for representing such business processes in a Web Services environment is WS-BPEL, the Business Message package will include one abstract BPEL process per interacting party (role) and each will include the business logic concerning the associated business role.

Note that the BPEL processes specified in the standard will be abstract BPEL processes which do not define links to a business' back-end system or show how the partner is supposed to process the documents it exchanges. Another possible set of files to include are default executable BPEL files for each party. These would include an

An Approach to Moving Industry Business Messaging Standards to Web Services Version 1.0

implementation of the abstract BPEL files that simply forward messages from and to the back-end systems of the partners as long as they are exposed as Web Services complying with certain specified portTypes.

The figures below show an example similar to RosettaNet PIP 3A4, which for simplicity have the acknowledgements, faults, and timeouts removed from the processes. This level of detail corresponds to the business view, which can be further refined by adding the acknowledgement, faults and timeouts to create processes. Figure 5-4 shows the abstract processes provided to a buyer and a seller. Figure 5-5 shows default executable BPEL processes that may be generated from the abstract ones. These then interact with the back-end at runtime.

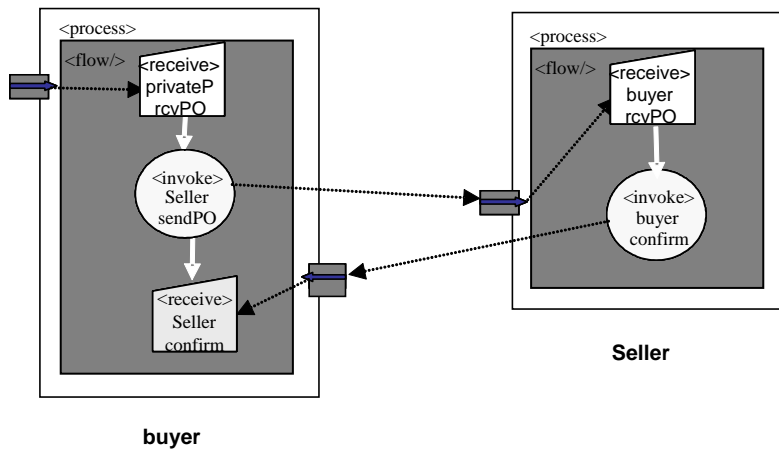


Figure 5-4: Abstract BPEL processes for each participant for submitting a PO. The rectangles with arrows indicate portTypes with operations. Timeouts and fault handlers are not shown.

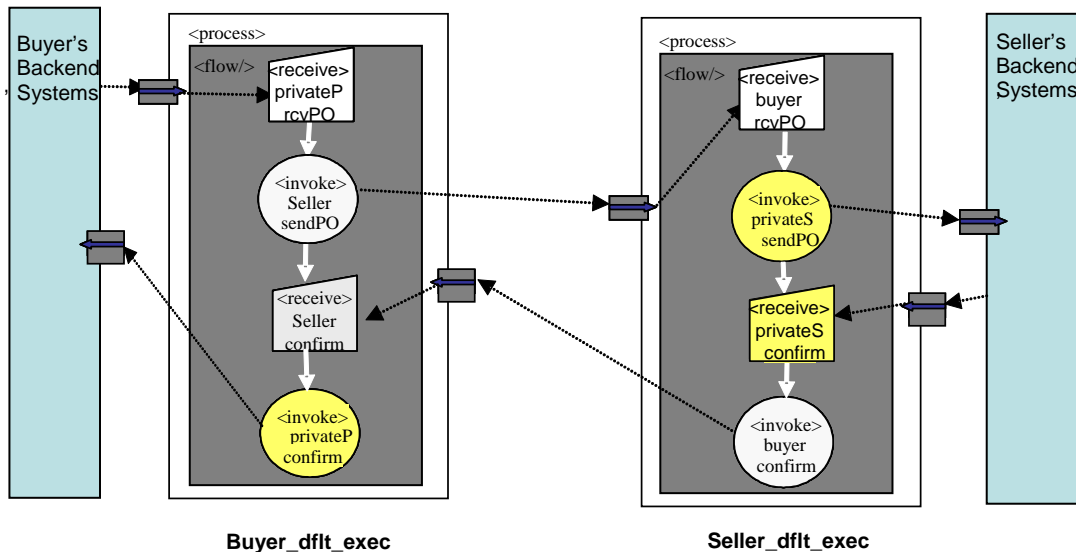


Figure 5-5: Executable BPEL processes that comply with the abstract ones above and interact with the back-end. New activities are in yellow. Timeouts, validation and fault handlers are not shown for simplicity.

BPEL provides the ability to manage message instances to ensure that messages corresponding to a specific business process instance do not accidentally get routed to a different one. It provides constructs to implement timing constraints and fault handling. BPEL processes can be designed to incorporate message validation and retries along with handling of duplicate, out of order, and post-completion messages.

For a more in-depth look at how BPEL can be used in the Web Services solution, please see Appendix A.

5.4 Define Web Services Quality of Service Using Policy Files

Quality of service requirements for Web Services is usually either defined within the given Web Services specification or via another closely coordinated specification. For our approach, we use policy assertions defined by the specifications to cover the various business operation parameters given in specifications. The policy assertions are based on the WS-Policy specification and attached to the appropriate WSDL operations per the WS-PolicyAttachment specification.

In particular, information related to security (authentication, integrity and confidentiality) will be specified using assertions defined in the WS-SecurityPolicy specification. Information related to reliable message delivery will also be included in the policy assertions defined in the WS-ReliableMessaging specification. Note that an assertion for *maximum number of retries* for message delivery is not yet defined in the WS-ReliableMessaging specification but we are hopeful that it will be added in the near future.

```
<wsse:Confidentiality wsp:Usage="wsp:Required">
  <wsse:Algorithm Type="wsse:AlgEncryption"
    URI="http://www.w3.org/2001/04/xmlenc#3des-cbc"/>
  <MessageParts>
    wsp:GetInfoSetForNode(wsp:GetBody())
  </MessageParts>
</wsse:Confidentiality>
<wsse:Integrity wsp:Usage="wsp:Required">
  <wsse:Algorithm Type="wsse:AlgEncryption"
    URI="http://www.w3.org/2001/04/xmlenc#3des-cbc" />
  <wsse:Algorithm Type="wsse:AlgCanonicalization"
    URI="http://www.w3.org/Signature/Drafts/xml-exc-c14n"/>
  <wsse:Algorithm Type="wsse:AlgSignature"
    URI="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <MessageParts Dialect="http://schemas.xmlsoap.org/2002/12/wsse#soap">
    S:Body
  </MessageParts>
</wsse:Integrity>
```

Figure 5-6: Part of a sample showing WS-Security assertions

Many of the remaining business operations parameters given in specification are built into the BPEL process. Time to acknowledge with a receipt signal, time to respond to an action, overall time to perform, and whether or not the time to respond to an action is included in the time to perform are all built into the timer functions defined in the BPEL process. Whether or not secure transport (SSL) is required is automatically taken care of by specifying HTTPS as the protocol in the URL for an operation's target endpoint.

However, there are two operational parameters that are not covered by any Web Services specification:

1. Is authorization required?
2. Is non-repudiation required?

A custom policy assertion can be created for "*Is authorization required?*" and consumed by a validation sub-system. For non-repudiation, the approach is not defined as yet. Section 6, Next Steps, cover this topic in more detail.

5.5 The Web Services Runtime Environment

The final part of this Business Messaging Web Services approach is the incorporation of a Web Services runtime stack during message transmission. Initially, only seven of the Web Service specifications are being considered:

- WS-BPEL
- WS-Security
- WS-ReliableMessaging

An Approach to Moving Industry Business Messaging Standards to Web Services Version 1.0

- WS-Addressing
- WS-Policy
- WS-PolicyAttachment
- WS-SecurityPolicy

The sub-system implementing WS-BPEL is responsible for running executable BPEL processes and maintaining their state. When receiving a message, the sub-system uses correlation sets to know whether to invoke a new process or route the message to a process that is already running. Note that WS-Policy assertions can be attached to BPEL constructs.

The sub-system implementing WS-Security performs signature and encryption processing for outbound messages and signature validation and decryption processing for inbound messages. For outbound messages, the sub-system should pick up processing properties and quality of service requirements from WS-Policy assertions associated with the WSDL operation. Inbound message processing should be automatic based on the presence of the WS-Security SOAP headers. Once non-repudiation support is added into WS-Security, it would also be used to handle sending back the requestor's signature and message digest with the business level acknowledgment message.

The sub-system implementing WS-ReliableMessaging provides reliable message transmission. It is only needed when transmitting or receiving specified action messages (per RNIF specification). For outbound messages, the sub-system is responsible for sending the message to the specified receiving party as many times as necessary until either an acknowledgment is received or a maximum retry threshold, if any, is reached. Processing properties and quality of service requirements should be picked up from WS-Policy assertions associated with the WSDL operation. For inbound messages, the sub-system is responsible for receiving the message and sending an acknowledgment back to the sender's WS-ReliableMessaging sub-system.

The WS-Addressing specification is either a current or future prerequisite for some of the other specifications (ie, WS-ReliableMessaging). In addition, addressing elements could be used by a BPEL process to allow Web Service target endpoints to be provided dynamically at runtime for each instance rather than only once at deployment for all instances.

Finally, the WS-Policy family of specifications is handled by a policy framework in a Web Services based system that can compute effective policies from the policy attachments, check their compatibility with the system's capabilities and the available services, and configure its subsystems to implement the relevant assertions (WS-Security and WS-ReliableMessaging sub-systems initially). While a specification is being developed for the dynamic exchange of policy files (WS-MetadataExchange), we do not mandate its implementation. At first, policy files may be exchanged manually or via some other mechanism.

**An Approach to Moving Industry Business Messaging Standards to Web Services
Version 1.0**

Note that the actual interplay between the sub-systems implementing the Web Service specifications is intentionally left out of the Web Services framework and will likely be vendor specific.

6 Next Steps

This paper represents a preliminary discussion on Web Services for Business Messaging. In developing this proposal, some functionality in some of the specifications that did not have a Web Services equivalent at this time were identified, assumptions were made, and some areas were left for future investigation. When the absent functionality needs to be provided, the assumptions need to be validated and any open issues need to be further explored and resolved.

Non-repudiation is an important piece for which a Web Services solution is just emerging but still awaits completion and standardization. Non-repudiation is a requirement for most legally committed business message handling and as such a necessity for business transactions. Non-repudiation support still needs to be discussed. However, until those additions are finalized, several other approaches may be taken for handling it:

- a. The abstract BPEL process can be defined to point to a presumed internal process that would then be implemented by the interacting parties as desired – thus permitting unique business relationships to undertake their own level of non-repudiation and relieving the burden from processes where it is not required.
- b. Individual participants can address their own internal business controls behind the WSDL and, as needed, expose their requirements in the WS-Policy layer.

The solution to mandate that the application middleware provide a sub-system for non-repudiation can be viewed as sidestepping the issue, but it has a better chance of success in the short term while the WS-Security updates are made. Interoperability with other non-repudiation sub-systems is not necessarily required, depending on the goals and complexity of the solution, and not all business messaging requires such due diligence. For now, our recommendation is that non-repudiation be provided by the middleware and offered in the standard as an optional process invoked by the participants, with the recommendation to moving to the WS-Security support once that becomes available.

Another area requiring further investigation involves accomplishing Consortium-specific processing. Some processing steps specified by RNIF are very specific to processing RosettaNet based messages and are not covered by the Web Services stack. In such cases where functionality is domain or consortium specific, one has the option of either defining domain-specific extensions to the given specifications or defining Web Services interfaces for possible consortium defined sub-systems that could do the relevant work and get called from the BPEL processes. One example is verification that the sender is authorized to participate in the given process. This requires knowledge of the sender and access to some kind of partner profile. Another example is additional dictionary validation for the content of some message elements. It may also be useful to have a standardized validation sub-system.

An Approach to Moving Industry Business Messaging Standards to Web Services Version 1.0

The following is a partial list of assumptions and open issues that need further attention:

- a. The WS-Security implementation must allow signature and encryption to be selectively applied to all elements of the SOAP message, including attachments. This allows full flexibility and adherence to current RNIF requirements. The WS-Security sub-system should also support WS-Policy / WS-SecurityPolicy.
- b. The WS-Security specification will need additional functionality for non-repudiation support.
- c. The WS-ReliableMessaging specification either needs to add a policy assertion for maximum number of retransmission attempts, or this maximum needs to be a standard feature of most WS-ReliableMessaging sub-systems.
- d. We do not address the use of *Debug Headers* in Business Messages.
- e. We do not address the persistence of Business Messages *per local policy* as specified by RNIF, except when related to non-repudiation.

Industry and the involved consortia themselves will need to eventually select and adopt its own solution set for its transition to Web Services. In fact, other standards bodies such as WS-I will likely be part of any solution set ensuring a high level of interoperability. For now, we offer a view into one possible approach. The membership of each consortium will ultimately drive the final solution.

7 References

CIDX

RosettaNet Implementation Framework: Core Specification Version: V02.00.01
Revised: 6 March 2002
<http://www.cidx.org/ChemeStandards/download.asp>

OAGi

RosettaNet Implementation Framework: Core Specification Version: V02.00.01
Revised: 6 March 2002
<http://www.openapplications.org/downloads/oagidownloads.htm>

PIDX

RosettaNet Implementation Framework: Core Specification Version: V02.00.01
Revised: 6 March 2002
<http://committees.api.org/business/pidx/xmlnew.html>

RosettaNet

RosettaNet Implementation Framework: Core Specification Version: V02.00.01
Revised: 6 March 2002
<http://www.rosettanel.org>

RNIF

RosettaNet Implementation Framework: Core Specification Version: V02.00.01
Revised: 6 March 2002
<http://www.rosettanel.org>

WS-Addressing

Web Services Addressing (WS-Addressing) August 2004
<http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/>

WS-BPEL (BPEL4WS)

Business Process Execution Language for Web Services Version 1.1 5 May 2003
<http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>

WS-I

Web Services Interoperability Organization is an open standards consortium chartered to promote interoperable Web Services. They provide specifications, profiles and reference materials to guide implementers of Web Services.
<http://www.ws-i.org/Documents.aspx>

An Approach to Moving Industry Business Messaging Standards to Web Services Version 1.0

Basic Profile 1.1 - Final Specification

Simple SOAP Binding Profile 1.0 - Final Specification

Attachments Profile 1.0 - Final Specification

Basic Profile 1.0 - Final Specification

Profile Conformance Framework WG Draft

Basic Security Profile Working Group Draft

WS-Policy

Web Services Policy Framework (WS-Policy) 28 May 2003

<http://www-106.ibm.com/developerworks/webservices/library/ws-polfram/>

WS-PolicyAssertions

Web Services Policy Assertions Language (WS-PolicyAssertions) 28 May 2003

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

WS-PolicyAttachment

Web Services Policy Attachment (WSPolicyAttachment) 28 May 2003

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

WS-ReliableMessaging

Web Services Reliable Messaging Protocol (WS-ReliableMessaging) March 2004

<ftp://www6.software.ibm.com/software/developer/library/ws-reliablemessaging200403.pdf>

WS-Security

Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)

OASIS Standard 200401, March 2004

<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>

WS-SecurityPolicy

Web Services Security Policy (WS-SecurityPolicy) Draft 18 December 2002

<http://www-106.ibm.com/developerworks/webservices/library/ws-secpol/>

WSDL - Web Services Description Language (WSDL) 1.1

W3C submission 15 March 2001

<http://www.w3.org/TR/wSDL>

XML Schema

XML Schema Part 1: Structures, W3C Recommendation 2 May 2001

<http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/>

XML Schema Part 2: Datatypes, W3C Recommendation 2 May 2001

<http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>

Appendix A The BPEL Solution In Depth

A.1 Basic BPEL

For an abstract BPEL process, we divide the entire definition into three parts: the definition of the *parties* the process will interact with, the definition of *process variables* to hold message values, and the definition of the *control logic* corresponding to the process flow. In BPEL, control logic is defined around a set of interaction activities that can invoke operations on other services and process invocations from other services. These activities are *invoke*, *receive* and *reply*. The first invokes operations on a partner's portType, and the last two receive the input to an operation offered by the process and send back a response message if one is required.

BPEL provides a set of partnerLinks to define the connections to the parties the process will interact with. PartnerLinks are instances of the partnerLinkTypes WSDL extension elements that must be added to the WSDL definitions. Each BPEL process created for a Business Message will contain one partnerLink for each instance of the partnerLinkType for the message. For simplicity the following examples use the RosettaNet 3A4 PIP Request Purchase Order for illustration, but CIDX, OAGIS or PIDX corresponding messages would be handled in a similar fashion. The partnerLink will specify which role the process plays and which role the partner will play.

```
<partnerLink name="seller" partnerLinkType="ns1:3a4PLT"  
  myRole="buyer" partnerRole="seller"/>
```

Figure A-1: Example BPEL partnerLink definitions

At least one more partnerLink (and partnerLinkType) needs to be created. BPEL requires that a business process provides at least one *receive* activity that can create an instance of the business process. The process that sends the first message will need to receive an external signal to start it. The other process will use the activity that receives the first message as its instance-creating *receive*. For the first process, therefore, we need to define a partnerLink to the arbitrary *starter* of the process.

BPEL variables are created to hold the messages the process will exchange. They are typed using the WSDL messages of the operations through which the messages will be exchanged. In abstract BPEL processes, a variable can be initiated with an opaque value. The opaque assignment signifies that compliant implementations must copy a value into the appropriate variable in any way they see fit.

```
<variables>
  <variable name="PO" messageType="ns1:PurchaseOrderRequestMessage"/>
  <variable name="confirm"
    messageType="ns1:PurchaseOrderConfirmationMessage"/>
  <variable name="ack" messageType="ns1:AckMessage"/>
  <variable name="faultInfo" messageType="ns1:faultMessage"/>
</variables>
```

Figure A-2: Example BPEL variable definition

BPEL enables the definition of arbitrarily complex control constructs using structured activities such as *sequence* and *flow* that include activities within them and dictate the order in which the included activities will execute. The appropriate top-level construct should be chosen when mapping to BPEL. In our experience, most Business Message specifications specify a straight sequencing of the message exchanges. Therefore a *sequence* activity or a simple *flow* with links may be chosen. To include forking in case of validation and the concurrency of receiving the acknowledgment or the second action message in a multiple action process, we start with a *flow* activity.

A process receiving a message corresponds to either a *receive* activity or the completion of an *invoke* activity. The *invoke* activity blocks until the response returns. A process sending a message corresponds to either a *reply* or an *invoke* activity. If the message is the response to a call to one of the process's request-response synchronous operations, then a *reply* is used. Otherwise, an *invoke* is used. These activities are used to choreograph the order of interactions in each business process. Note that the BPEL processes being created are mirror images of each other, such that an *invoke* in one process corresponds to a *receive* on the other. Given the above information regarding the partners, the variables, and the operations provided, one can easily map from the Business Message specification to the BPEL *receive / invoke / reply* activities in a *sequence* activity. Next, one can add the additional information such as timeouts and failure notifications.

```
<flow>
  <links> <link name="l1"/> <link name="l2"/> ... </links>
  <receive name="begin" partnerLink="buyerPrivate" variable="PO"
    portType="ns1:PIP3A4ReqServicePort" createInstance="yes"
    operation="PurchaseOrderRequest">
    <source linkName="l1"/>
    <correlations>
      <correlation set="RN_IDs" initiate="yes"/>
    </correlations>
  </receive>
  <invoke name="sendPO" partnerLink="seller"
    portType="ns1:PIP3A4ReqServicePort"
    operation="PurchaseOrderRequest" inputVariable="PO">
    <target linkName="l1"/>
    <source linkName="l2"/>s
  </invoke>
```

Figure A-3: Example BPEL control logic

A.2 Managing Message Instances

In BPEL, a message that comes to a *receive* activity with a *createInstance="yes"* attribute will create an instance of the BPEL process, corresponding to an instance of the business process. Subsequent messages targeted at that instance will be routed to the same BPEL process instance. BPEL defines a correlation mechanism to enable it to maintain conversations over the lifetime of a process instance. Named, typed properties are defined and aliased to different parts of the WSDL messages used by the process. These properties may then be grouped into correlation sets and attached onto the interaction activities.

Using correlation enables one to guarantee that messages corresponding to a specific business process instance do not accidentally get routed to a different one, while still enabling multiple instances to be run concurrently.

```
<correlationSets>
  <correlationSet name="RN_IDs" properties="ns1:PIP_ID ns1:PIP_InstanceID"/>
</correlationSets>
```

Figure A-4: Example BPEL correlation set

In the case of mapping RosettaNet PIPs to Web Services in the accompanying example, we propose using the Message Code and the Message Instance ID as values to correlate on. In order to ensure uniqueness, it is possible that information about the trading partner will also need to be included in the correlation set. This needs further evaluation.

A.3 Timeouts

BPEL contains constructs for the definition of delays, alarms and fault handling that can be used in order to represent the timing constraints on certain aspects of the process, as well as to specify the detection and notification of failures.

To specify that a message must be received within a certain time limit, one surrounds the receive activity in a scope with an alarm handler. The activity in the alarm handler will be executed if the alarm goes off. However, if the receive completes before the alarm goes off, then the alarm will be disabled and processing will continue regularly. This approach is used to create the BPEL behavior specified in the timing constraints of the process.

```
<scope>
  <eventHandlers>
    <onAlarm for="PT8H">
      <sequence>
        <assign> ... </assign>
        <invoke name="timeout1"
          partnerLink="seller" portType="ns1:PIP3A4ReqServicePort"
          operation="fail" inputVariable="faultInfo"/>
        <throw faultName="ns1:genericFault" faultVariable="faultInfo"/>
      </sequence>
    </onAlarm>
  </eventHandlers>
  <receive name="sellerAck" partnerLink="seller"
    portType="ns1:PIP3A4ConfServicePort"
    operation="ReceiptAcknowledgment" variable="ack">
    <correlations> ... </correlations>
  </receive>
</scope>
```

Figure A-5: Example BPEL timer logic

In the case of this RosettaNet example, if no message is received in the allocated time, an operation should be invoked to inform the other party of the failure, and a fault should be thrown in the process itself. If not caught, such a fault would terminate a process. Throwing a fault, instead of using *<terminate>* to end the process, enables compliant

implementations of the abstract process to catch such a fault and perform any necessary clean-up needed, before termination. If this is the result of a timeout then the case of RosettaNet, the specification requires that an error message be invoked (OA1), which, for example, the owner of the process may want to send as a failure message to their own back-end systems. The other standards have similar message specifications.

A.4 Retries

Retries may be modeled in BPEL using a combination of *while* loops, a pick activity that can be triggered based on either a message or a timeout, and links. On the receiving process, a scope that waits for the maximum time would be created to handle duplicate messages and resend the acknowledgements as required. However, this clutters the business logic with an attribute that really belongs either to the lower level networking layer, if it is regarding resending one message, or to a higher level workflow management system layer if it is regarding restarting the entire process or subsections of a process.

Our recommendation will rely on the WS-ReliableMessaging sub-system to ensure reliable message delivery. It will perform all necessary retries within its context.

A.5 Validating Messages

Web Services require that transported messages conform to the declared schemas. As such, Web Services implementations perform schema type checking on the incoming messages before they reach the underlying service implementations. These checks represent the first minimal level of business validation. In such cases there is no need for a representation of the validation in the business process.

There are situations where schema checking is not enough and a business exchange requires that further dictionary or business relationship validation be performed. In such cases, one may represent this in the BPEL process by defining an *assign* activity that places an opaque token into a boolean variable. The boolean variable is then checked. If it evaluates to false then a fault is thrown. This could be used to indicate to compliant implementations (possibly executable BPEL processes) that they must perform the additional check and place its result in the variable.

Note that if links are used, then the *suppressJoinFailure* attribute should be set to *yes* for this setup. Otherwise, a link with a fault condition could terminate the entire process instead of our expected behavior of simply skipping the target activity.

A.6 Faults

In addition to the fault defined due to timeouts, business signals set to signal failure are also modeled by invoking an operation provided by the trading partner that can consume an error message. On the receiving side, the BPEL process would contain an event handler that can receive such exception messages from its partner(s). The event handler's activity would contain the reaction that the process is supposed to have upon failure notification.

Another kind of fault notification is present in some specifications. An example is using the OA1 Notification of Failure PIP or the Confirmation BOD in OAGIS. The general guideline is that this type of message will be used if it is possible that the partner is to be notified of a process that is no longer executing. Since such messages are unique in their own right, they will have their own WSDL and BPEL files and a corresponding partnerLink in the BPEL process. If the original process needs to initiate a notification of failure, it would invoke the operation offered by that message's portType. If BPEL executable processes are used in combination with the failure messages, then one must provide some way to propagate the fault to the process in question. How this is done is left out of RNIF and we therefore relegate it to the private business logic/systems.

Note that RNIF recommends that OA1 PIP messages be transmitted via a different channel than regular messages. E-mail, or perhaps telephone or pager messages, would be possible alternatives. In either case, an administrator on the receiving side will need to evaluate the failure condition and act appropriately. Enterprise level workflow management systems usually provide administrative capabilities to monitor and manage running processes, which can be used to terminate the corresponding processes. Alternatively, if one uses a lightweight system, the executable BPEL process could provide an event handler for a local administrator to terminate the process.

A.7 Duplicate, Out-of-Order, and Post-Completion Messages

The message choreography and handling requirements specified by Business Messages and incorporated within the RNIF or other frameworks that use asynchronous messaging can sometimes result in receiving duplicate messages, receiving messages out of expected order, and possibly receiving messages after the associated process has completed. RNIF does not address messages beyond a completed process nor does it address multiple PIPs running concurrently, which can natively be handled by executable BPEL.

A duplicate action message may be generated if the sending party does not receive the associated receipt acknowledgment or exception signal message from the intended receiving party, within the required time limit. With reliable message delivery now handled by the WS-ReliableMessaging sub-system, using a delivery assurance of *Exactly Once* should eliminate duplicate messages getting through to the BPEL process.

An Approach to Moving Industry Business Messaging Standards to Web Services Version 1.0

Given the nature of asynchronous messaging and parallel processing, it is possible that a response action message may arrive before the receipt acknowledgment signal message associated with the initial action message. This situation is handled by the BPEL process via a *flow* structure that allows for concurrent behavior, or a forking of the process.

With the current message handling process described by RNIF, there are at least two scenarios where a message may arrive after the process has completed – one for a single-action message and one for a dual-action. For example, in the scenario for a single-action message, Trading Partner A sends an action message to Trading Partner B and then waits for the receipt acknowledgment / exception signal message. If Trading Partner B received the action message successfully, it will send the receipt acknowledgment back to Trading Partner A, process the action, and then end. If Trading Partner A never receives the receipt acknowledgment and Trading Partner B's process ends before A sends the action message again, then there will be no process running at B to consume the duplicate action message. Fortunately, since this is a duplicate message situation, the message will be stopped by the WS-ReliableMessaging sub-system as described above.

With all the pieces in place, we now depict an example abstract process in figure A-6 below again using the RosettaNet PIP 3A4 Purchase Order Request as the example.

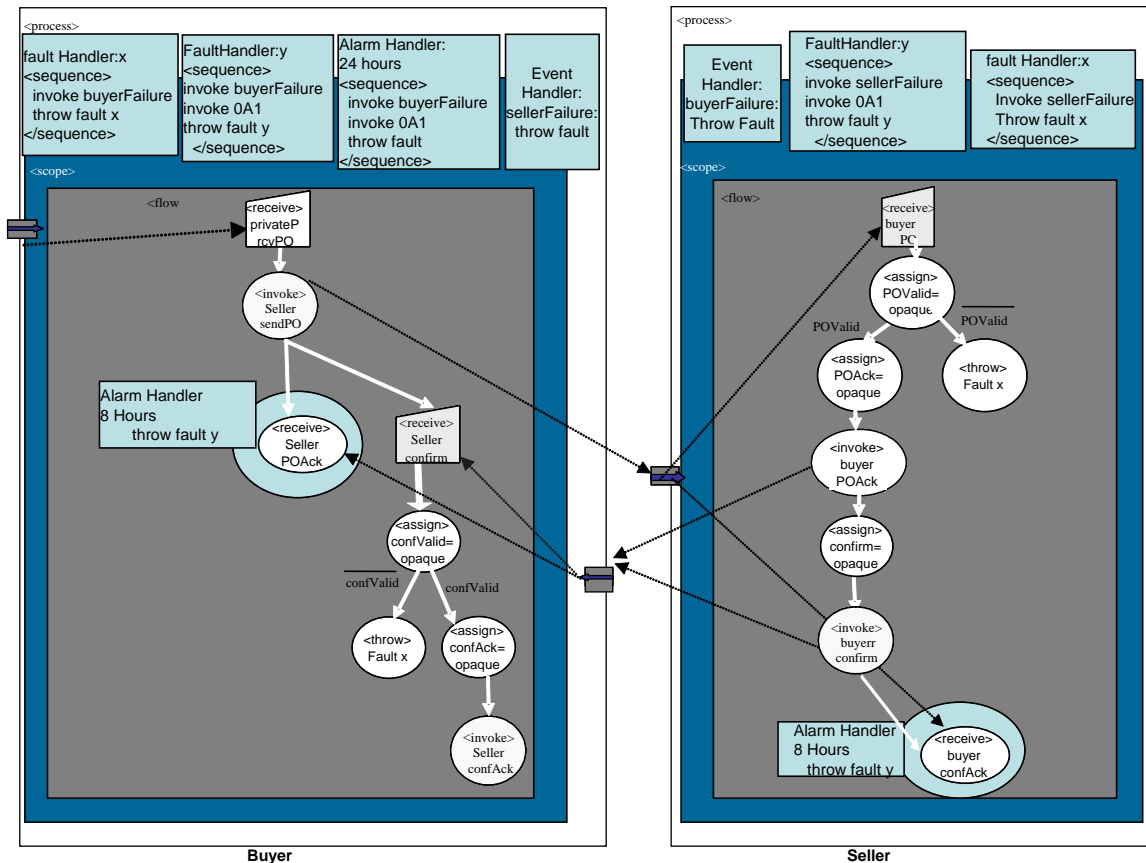


Figure A-6: Example BPEL abstract processes for PIP 3A4 Purchase Order Request

A.8 Default Executable BPEL Processes

It is possible to derive default executable BPEL processes to be included in the Web Services Business Messaging package. For each abstract process, a default executable can be created with the assumption that the back-end systems of the party that wants to use them complies with the provided WSDL portType. It is important to note that this is an extra capability that provides one possible execution variant. The parties may choose to implement the abstract BPELs using other means and are not forced to use executable BPEL.

BPEL itself does not formally specify what it means for an executable process to comply with an abstract one. The restriction we follow in such creation is that the projection of each executable process relating to the partner defined in the Business Message specification is not violated. In other words, other partners can be added and interactions with them interleaved in the process definition. It is disallowed, however, to modify existing interactions with any partners already defined in the abstract process.

The executable BPEL will add all necessary logic required by that partner to comply with and execute its end of the multi-party interaction. In particular, this includes interacting with the company's back-end systems for such functionality as procurement and inventory. The company may decide to either expose each of its subsystems as Web Services or provide one Web Service that overlays its IT infrastructure. In either case, the executable process can interact with them.

One approach for deriving these processes is to add to each a partnerLink representing the company's systems and forwarding all incoming messages to those. On the other hand, any variables that are opaquely assigned in the abstract process are then initialized either as the result of the response to an *invoke* or by a *receive* activity that receives a messages containing the values of those variables from the back-end systems. In one demo set-up, the back-end systems consisted of forms that take human input, Web Services clients, and services that share access to data in the file system.

A global fault handler is also added that notifies the back-end systems of any faults raised and not handled that could cause the process to terminate.

The implementations of the back-end systems are not part of the executable process. It only cares that it can forward the incoming messages to them and that it can get any data it needs from them. The main thing to take away is that this approach presents only one way to create a compliant implementation. Any additional or specialized adaptation steps necessary for interacting with a company's back-end systems may be used and any allowed customization to the process may be performed.