bei/2004-01-02

Date: January 12th 2004

Version: 1.0.2

Revised Submission to BEI RFP bei/2003-01-06

Business Process Definition Metamodel

Submitters:

International Business Machines

Adaptive

Borland

Data Access Technologies

EDS

88 Solutions

Supporters:

BEA Systems

BPMI.org

Unisys

Part I – Introduction

Table of contents

COPYRIGHT WAIVER

NOTICE

Submission Contacts

The primary contact person for this submission is Sridhar Iyengar (IBM), any feedback on this submission can be sent to any of the submission contacts.

IBM

Jim Amsden
Joachim Frank
Tracy Gardner
Pablo Irassar
Sridhar Iyengar (siyengar@us.ibm.com)
Simon Johnston
Stephen White

Adaptive

Pete Rivett (pete.rivett@adaptive.com)

BEA Systems

Ed Cobb (ecobb@bea.com)
Yaron Goland

Borland

Karl Frank (Karl.Frank@borland.com)
Randy Miller

BPMI

Layna Fischer (fischer@bpmi.org)

Data Access Technologies

Cory B. Casanave (cory-c@enterprise-component.com)

EDS

Fred Cummins (fred.cummins@eds.com)

Unisys

Sumeet Malhotra (sumeet.malhotra@unisys.com)

88 Solutions

Manfred Koethe (koethe@88solutions.com)

Overview of Submission

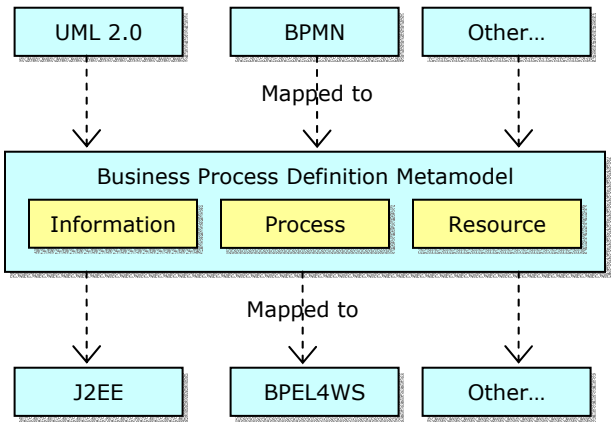
This document provides an more detailed response to the Business Process Definition Metamodel Request for Proposals. This is the first revised submission, and although far more complete than the initial submission it is expected that a further revision will be needed to complete the definition.

In addition to responses to each of the requirements set out in the RFP, a description of the objectives of the submission is included to further specify the intent of this submission.

The primary objective of this submission is to provide an abstract model for the definition of business processes either as a stand-alone activity or in the context of a broader business modeling program. This metamodel is positioned both in terms of its role within the broader context of business modeling as well as in the context of the OMG Model Driven Architecture (MDA) initiative.

We also provide appendices that demonstrate how this metamodel may be used by tools with alternative graphical notations (by mapping the Business Process Modeling Notation, or BPMN, to this metamodel). The UML 2.0 is a perfectly adequate metamodel for capturing business models and specifically business process definitions; however it is clear that only a subset of UML2 used in particular ways is useful and so our work is in the form of a constrained subset of the UML. We expect that future revisions will expand this support by encompassing the work of existing business modeling notations as well as demonstrating the use of the UML notation for business modeling.

We also demonstrate how this metamodel can be used to generate platform-specific process languages, specifically the Business Process Execution Language for Web Services (BPEL4WS). These mappings act as a validation of the model by ensuring that metamodel fits the needs of business modelers, and that developed models can be automated and deployed.



Finally, the UML profile introduced in the submission is documented and demonstrated by example; however more detail will be provided in a revised submission.

Guide to the material in this submission

The metamodel documented in this submission provides a common language, for describing business processes in an implementation independent manner. This is not to say that the models are abstract from execution details, on the contrary it is our aim to describe executable processes, these models are intended to be abstract from the detailed implementation (platform) mechanisms. This shared subset of UML 2.0 also acts as an agreed base for translation to such implementations where required. The adopted UML 2.0 superstructure metamodel provides much of what is required for the definition of business processes.

The following semantics are not present in UML 2.0 and are introduced in the Business Process Definition Metamodel:

- Resource modeling (deferred until later submission)
- Access control (deferred until later submission)
- Compensation
- Annotations for simulation (deferred until later submission)
- Correlation (routing of incoming requests to the appropriate process instance, deferred until later submission)
- Observation (audit/monitoring, deferred until later submission)

The specification of the exact UML 2.0 subset in the area of PIM and alternate PSM models is still under development but will be completed in a revised submission.

The submission contains the following specification details.

UML 2.0 Profile for BPD

The metamodel proposed here is specified as a UML 2.0 profile because, although a specialized notation for Business Process Definition may be preferable for some scenarios, there is also value in enabling generic UML tools to both author and consume business models. It will be important for some of the identified roles because they would be transforming from the business models into IT models, such as transformations from business information models into data models.

UML 1.5 Profile for BPEL4WS

An existing UML 1.5 profile for automated business processes with a mapping to BPEL4WS is included in Appendix A. This profile will be updated to meet the requirements of this RFP. The mapping has been implemented and provides a proof of concept for mapping from a UML-based business process definition language to a specific execution language.

The audience for this model is primarily technical information technology staff realizing processes in runtime software. The profile acts as a UML-based visualization for a language which has no standardized or described graphical notation.

Mapping from BPD Metamodel to BPEL4WS

This identifies a mapping from this profile directly to BPEL4WS.

The audience for the BPD metamodel is business users more concerned with the capture of the business semantics of their processes than the details of a runtime implementation. This mapping provides a route from the business-level model directly to a runtime model.

Mapping from EDOC to BPD Metamodel

This demonstrates that the set of concepts and model elements developed by EDOC are similarly available when using the BPD Metamodel. In this case by mapping from the EDOC profiles to the BPD profile an automated mapping could be developed by a tool vendor to translate user developed models from EDOC to BPD.

Mapping from BPMN to BPD Metamodel

It is also important to understand that the UML notation however is frequently seen as daunting or too "technical" for business analysts and so we expect and encourage others to provide alternate notations for the subset of the UML that we propose here.

One of the submission goals is to make the UML notation less daunting to the business user, primarily by understanding existing notations. This will be further addressed in later submissions.

In the case of the BPD metamodel we will provide an example mapping from the BPMN notation to the subset of UML we have identified. This would provide tool vendors the opportunity to support users requiring a BPMN graphical notation using the standardized semantic metamodel we present here.

Overall design rationale

This submission does not require any changes to the UML 2.0 metamodel. Where possible the submission does not introduce new elements to the UML language, but identifies a subset of the UML that can be used to describe business process models. However, where concepts need to be disambiguated a profile is provided to introduce a small set of stereotypes.

This metamodel does address automated business processes, through transformations to process automation languages such as BPEL4WS. It is also the goal of this profile to model processes that are not intended for automation in this way – processes that are solely implemented through collaborating people.

Statement of proof of concept

Substantial parts of the metamodel presented here are based directly on the metamodel underlying products in development at IBM. WebSphere Business Integration (WBI)

Modeler[1] is a platform-independent business process modeling tool that also provides a transformation to a runtime language (MQSeries Workflow FDML).

IBM is in the process of updating this product line and at the same time integrating with the product lines from the Rational brand. This update includes a new metamodel, moving from a proprietary base to one based directly upon UML2 and including the experience from Rational in applying the UML to business modeling. This submission describes a subset of the metamodel for that product, some areas are altered to reflect the fact that only a subset has been taken and some associations are therefore not available.

In the area of translation to automated business process languages with the mapping from UML to BPEL4WS; IBM has made a version of tools to implement this mapping available on the IBM alphaWorks web site[2]. This technology can generate BPEL4WS from UML models in either IBM Rational Rose or IBM Rational XDE. In the future this mapping will be used to define the transformation of models from the platform-independent WBI Modeler into BPEL4WS.

Change History

Revision	Date	Comments
1.0.2	January 2004	Revised Submission. • Included new co-submitters including BPMI.org. • Reorganized to better follow the standard submission structure. • Updated references and included the BR SIG business modeling white paper. • Introduced the stereotype «Document» to the profile, describing its relation to «Entity».
0.9.1	December 2003	Working Group draft for Revised Submission. Added Unisys as supporter, correctly placed BEA as supporter rather than submitter. Added introductory section on principles. Renamed the modeling view "Policy" to be "Governance" in line with the expanded definition in the IBM Business Rules Submission. Process and Entity now use the meta class "Class" (StructuredClass) instead of Component. Updated the diagrams in the examples. Added initial section on organization. Included Appendix on mapping from EDOC to BPD. Included updated BPMN mapping.
0.9	August 2003	Initial Submission

[1] http://www.ibm.com/software/integration/wbimodeler/

[2] http://www.alphaworks.ibm.com/tech/ettk

Response to RFP Requirements

Mandatory Requirements

Mandatory Requirement	Resolution
6.5.1 Required Metamodel *Responses to this RFP shall provide a metamodel forming an abstract language for the expression of business process definitions.* *Proposals shall depict the solicited metamodel using UML.*	This proposal reuses a subset of the UML 2.0 metamodel and extends UML 2.0 with a Business Process Definition package. UML 1.5 and UML 2.0 profiles of the metamodel are also provided to enable to metamodel to be used with current and future vanilla UML tools.
6.5.2 Metamodel Compatibility Proposals shall use the appropriate elements of existing metamodels including, UML, EDOC, MOF and OCL.	We observe that much of the modeling capability introduced in EDOC is now directly supported in UML 2.0. This proposal therefore builds directly on UML 2.0. OCL can be used with this proposal wherever a side-effect free expression language is required. We observe that there is an overlap between the concepts required to fully specify executable business processes and the concepts required for service-oriented architecture. We recommend that SOA is covered in a separate RFP which is compatible with this one. An appendix to this document describes the relationship between this specification and EDOC.
6.5.3 MOF Compliance The resulting metamodel shall be MOF-compliant.	The metamodel introduced by this proposal is MOF-compliant as it is currently a subset and profile of UML2 and not a separate metamodel.
6.5.4 Procedural and rule-based flow of control Proposals shall provide for specification of process flow based on control flow from completed activities to activities to be initiated as well as initiation of activities based on rules or pre-conditions. Comment: We assume this means ActivityNode in the UML 2.0 sense	This proposal supports procedural control flow by reusing the activity and action modeling elements of UML 2.0. This proposal will support the initiation of control based on an expression becoming true through UML 2.0 AcceptEventActions with ChangeTriggers and TimeTriggers. The latter provides the ability to initiate control based on policies such as "start this process at 11:00am on a Friday".

Mandatory Requirement	Resolution
rather than Activity and is therefore referring to control flow within a process. Comment: The term precondition should not be used here as it refers to something that is not side-effect free.	
6.5.5 Specification of Activity Performers Proposals shall provide for the specification of selection criteria for performers and resources, including human performers, applications, passive resources and sub-processes., *The basis for selection shall include* *a) Specific resource identity* *b) Resource attributes* *c) Relationships with other assigned resources* *d) Relationships to subject matter* e) *Combinations of the above.*	Tasks may have associated resource requirements which are expressed in OCL. Yes. Resource types are classifiers that have instances with identity. Instance specifications pinpointing a particular resource may be associated with a task. Yes. Resources can have attributes. Yes. The deployed resources of a process will be accessible to resource queries. Example will be included Yes. Resource requirements can refer to process variables, task input objects and constants. Example will be included Yes.
6.5.6 Asynchronous and Concurrent Execution Proposals shall provide mechanisms for specifying concurrent execution of activities: *a) A process model shall be able to define when multiple, concurrent activities are initiated.* *b) A process model shall be able to define when an activity or completion of a process depend on the completion of multiple, concurrent activities. This shall include initiation of an activity based on completion of n of m concurrent activities (where n <= m).* *c) Business processes shall be able to invoke processes that execute asynchronously.*	Yes. The control flow semantics of UML 2.0 activities are adopted by the profile. UML 2.0 join specs provide the abilities to specify complex joins within a process instance. This is the normal case with the implicit creation semantics adopted by this proposal. The case where a business process is instantiated with a synchronous call and completes on return is simply a special case. Synchronous and asynchronous sending of messages is supported as in UML 2.0. Point-to-point and broadcast semantics are supported. Asynchronous receipt of messages/events is supported as in UML 2.0.

Mandatory Requirement	Resolution
d) The model shall support specification of the publication of events and messages for asynchronous delivery. e) *The model shall support receipt of messages from a collaborator and subscription to events. Messages and events shall be received as asynchronous inputs to a receiving activity executing concurrently with other activities in the process.*	
6.5.7 Specification of initiation and termination *Proposals shall provide modeling constructs for specifying when and how activities and processes can be initiated and terminated:* *a) Pre or post conditions* *Comment:* Preconditions and postconditions in UML do not alter runtime semantics. Perhaps this is intended to allow 'when conditions' that emit a token when an expression becomes true. Actions for initialization and termination with consideration of actions required for abnormal termination, including the initiation of compensating processes. *b) Propagation of termination to active activities and sub-processes.*	Initiation and termination of processes is implicit rather than explicit from the point of view of a client of the process. This decouples the service requester and service provider and facilitates scenarios where the first of a number of events will lead to a new instance with the subsequent events being routed to the existing instance. *Routing events to existing instances requires correlation, which will be introduced in extension of UML 2.0*. Routing not currently described in this revision a) Supported through UML 2.0 AcceptEventActions with ChangeTriggers. b) Initialisation of processes and tasks is implicit. Exception handling is supported as defined in UML 2.0. UML 2.0 does not include compensation. This proposal adopts the BPEL4WS semantics for compensation within a process. c) UML 2.0 ActivityFinalNodes and Exception semantics are supported. The notion of subprocesses requires a notion of composition which is provided through UML 2.0 composite structures. Note, we differentiate between embedded sub-processes (StructuredActivityNode) and invoked/shareable sub-process which are Activities.
6.5.8 Choreography Proposals shall provide for the specification of the signals and	Supported. Choreography is expressed using collaborations with roles. The types of the roles specify the interfaces provided and required by the roles. Permissible choreographies are

Mandatory Requirement	Resolution
exchanges performed between processes to achieve collaboration as described in 6.2.5.	specified using activity graphs. Example to be included, specifically to demonstrate the notion of global processes
6.5.9 Audit Log Generation *Proposals shall include provisions in the metamodel to allow the specification of business logic related audit log records. This part of the metamodel shall provide for the specification of:* *a) The content of the log record in relation to the process definition* *b) The timing of the log record emission*	Supported. Audit log message types must include identification and time properties. An audit action adds identification and time information and sends such events to a logical audit service. Audit messages can contain properties that will be populated with expressions that have access to process variables. A MOF identifier is included in the log to enable the event to be interpreted in the context of the business process definition. Information to identify the process instance is also added. (Details TBD.) The audit action automatically adds the current time to audit messages. This is not currently described in this revision
6.5.10 Distributed Execution Proposals shall ensure that the form of definition does not preclude distributed execution.	Supported. No assumption is made that partners (including sub-components) are in the same namespace or at the same physical location. No support is provided for distributing an individual process instance – sub-components should be introduced where this is required.
6.5.11 Process Definition import and export Proposals shall support XMI export and import for exchange of process definitions.	Supported. XMI is the default import/export format. At this time there are no metamodel extensions and so standard XMI should be sufficient.
6.5.12 Non-Normative Notation Mappings *As a proof of concept, proposals shall provide non-normative mappings to two recognized business process modeling languages, e.g.:* *BPEL4WS* *XPDL* *(See section 6.4 for a non-exclusive list of references to relevant specifications)*	A mapping to BPEL4WS is provided. Reverse mappings from one or more 'legacy' workflow languages will also be included. Mappings to other execution languages can be defined as required.
6.5.13 Compatible Versions of	Yes. The final, revised submission will be based

Mandatory Requirement	Resolution
Existing Specifications The final, revised submission shall be based on the most recently adopted version of related specifications (e.g., UML and MOF) that is adopted three months prior to the final revised submission deadline for this RFP.	on the most recently adopted versions of standards, as required. A profile for UML 1.x will also be provided as the marketplace will not move completely to UML 2.0 immediately.

Optional Requirements

Optional Requirement	Resolution
6.6.1 Additional Non-Normative Mappings Proposals may provide additional mappings to recognized languages for business process definition, complementing the list of mandatory mappings requested in 6.5.12.	To be decided.
6.6.2 Additional Execution Constraints *Proposals may provide for the ability to model additional execution constraints, like maximum duration of a process or activity execution. For these additional constraints the behavior of constraint violation should be modeled and its effect on the process enactment described.*	None identified at this stage.

Issues to be discussed

Issue to be discussed	Discussion
6.7.1 Relationship to existing UML metamodel *Proposals shall discuss the relationship of model elements used for business process modeling to the existing UML metamodel to demonstrate consistency with the UML metamodel.*	This proposal reuses the modeling elements introduced in UML 2.0. Where appropriate concepts are not directly supported extensions in keeping with UML 2.0 are introduced. UML 2.0 Uses Cases may be used for documenting the external view and business value of business processes. UML 2.0 Activities and Actions are used for the business process definitions and

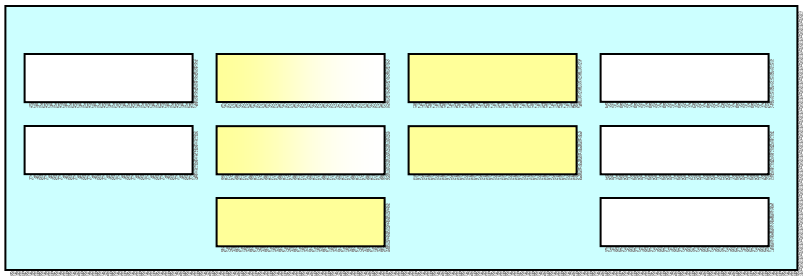
	choreographies. UML 2.0 Collaborations and Roles are used to specify interaction patterns between partners. UML 2.0 Templates are used to permit the use of templates in business process definitions. UML 2.0 Composite Structures and Activities are used to represent composite business processes.
6.7.2 Relationship to Related UML Profiles, Metamodels and Notations Proposals shall discuss how business process definitions may be incorporated with or complement other UML profiles, metamodels and notations for specification of business systems, particularly the UML Profile for EDOC.	A mapping from EDOC to BPD is included as an appendix.
6.7.3 Mapping to Existing Business Process notations and UML Notation Proposals shall discuss how the proposed metamodel may be mapped to existing process definition notations as a demonstration of completeness and compatibility.	Currently the submission includes a mapping from an existing business modelling notation (BPMN) to the metamodel defined. UML 1.5 and UML 2.0 profiles corresponding to the BPD metamodel are a part of this submission. Instances of the profile can be expressed using UML Notation with stereotype keywords or icons. While a tailored notation may be preferable, this option is available for creating business process definitions using generic UML 1.5 and UML 2.0 tools.
6.7.4 Resource Model Proposals shall describe assumptions regarding an associated resource model.	An associated resource model must provide the domain for resource requirement specifications (6.5.5). We will propose a resource model for the definition of individual, bulk, resource types, and roles, which represent a resource's capability to perform tasks. Roles may be scoped, to account for the dependency of such capabilities on subject matter. This is not currently described in this revision
6.7.5 Relationships with related OMG specification activities. Proposals shall discuss how the specifications relate to the specification development efforts currently under way as noted in Section 6.4.3.	UML 2.0 Covered above. Business Process Runtime Interfaces MOF 2.0 Versioning MOF 2.0 Queries/Views/Transformation MOF 2.0 Q/V/T is under definition at the time of writing. The final expression of mappings from this profile to execution languages will use the QVT formalism assuming that QVT is

	sufficiently complete at that time. CWM Provides a detailed metamodel for describing data and information, this should be supported as an alternate way to describe BPD information models. There is an existing Organizational Structure RFP that should supersede the organization model presented in this submission. The Business Rules RFPs should be able to describe constraints on processes models.
6.7.6 Consistency checks *Proposals shall discuss how the specification supports consistency checks, particularly between choreography specifications and a business process that participates in the choreography.*	The proposal defines the conditions under which an executable process can be said to realize a role in a choreography.
6.7.7 Access Control Proposals shall discuss how access authorization for process data, artifacts, activities in a process, and process enactment may be based on process roles of individuals associated with a specific process instance.	Supported. Access control will be consistent with Resource specification.
6.7.8 Web services and collaboration support Proposals shall discuss how the specification supports the definition of business processes and choreography for web services and other collaborations including the relationships with messages, documents, interface specifications, participant roles, signatures and message exchanges.	This proposal adopts an approach that is consistent with a service-oriented architecture and can therefore be mapped to web services technologies. This proposal will not provide a UML profile for service-oriented architecture but will introduce modeling elements as required. If a standard SOA profile is available in a suitable timeframe then this proposal will depend on it.

Part II – Submission Details

Scope of the submission

This submission looks at the problem of business process definition in context. In the following sections we will describe the problem of business process definition as a part of a broader activity – business modeling. Business modeling is, as the diagram below demonstrates, comprised of a set of related disciplines. In many cases an organization is only focused on certain of these areas, for example business strategy is an area of particular interest right now.



In terms of this submission the scope is restricted to the areas of Process, Automation and Resources. However it is clear that no process model can be complete without addressing information and organization and this submission will present default models in this area.

- ***Strategy***; encompasses business goals, strategic objectives and can be used to measure business performance. Note, not in the scope of this submission.
- **Governance**; includes the modeling of regulatory concerns, policies and rules.
- ***Information***; high-level modeling of information sources and includes modeling of business artifacts, such as documents, products, parts, assemblies, etc.
- ***Organization***; the ability to capture organizational structure. An organization's responsibility (for tasks) and ownership (of resources) can only be specified in conjunction with other models (Resource, Process). The main purpose of an Organization Model is to define organizational units (such as companies, project teams, departments, etc.) and their interrelationships. With this in place, Resources, Tasks, etc. can then be referenced, and the underlying semantics may defined as ownership of, responsibility for, etc.
- ***Resources***; a refinement of the information and organizational models that provides the notion of a resource, resource cost, ownership, participation, availability etc. Most importantly: Resources have "qualifications", defined as the set of roles they can fulfill in a process. This is the main property differentiating a "Human Resource" from a "Person", for example.
- ***Process***; modeling (at higher and lower levels of detail) the activities of a process.
- ***Automation***; the platform-specific modeling of an automated process, capturing details defined by a platform model for the execution environment.

- ***Simulation***; in effect a platform model, in that it defines an abstract execution environment for the purpose of validating and refining the automation model. Note, not in the scope of this submission.
- ***Financial***; expands on the resource model and compliments the simulation model to capture financial measures. Note, not in the scope of this submission.
- ***Observation***; the capture of points in a process where information should be logged, or measures should be taken to monitor and report process performance. This can be accomplished using the auditing capabilities of this submission.

As we have stated we intend this submission to provide an implementation independent model for capturing business process; in MDA terms this is a platform independent, or PIM, model. In the same regard we can see automated process languages such as BPEL4WS as platforms and therefore the UML profile provided in the submission can be described as a platform specific, or PSM, model.

Such a review of the requirements to support an end-to-end business modeling discipline would not be complete (or even meaningful) without a review of the users and roles that we need to support. It is important, not just in understanding the concepts that the metamodel must support but also from a notation and tool implementation point of view to understand the user's needs and expectations.

- ***Business Analyst*** – responsible for capturing the state of the business, analyzing its operations and recommending improvements that add value to the business or the customer.
- ***Process Specialist*** – has detailed knowledge of a process, its current nature and also how it fits into the overall business strategy.
- ***Solution Architect*** – responsible for ensuring that business systems support the goals of the business by implementing the defined processes in a cost effective manner and respond to changes in the process definitions in a timely fashion.
- ***End User*** – stakeholders in core processes and certainly affected by any change to the organization or processes of the business. We can expect them to be consumers of organization and process models, specify certain processes, and a good check and balance in the constant pace of change.
- ***Executives*** – are a key part of the process of reviewing the models created of the business. Specifically they are direct stakeholders of the business strategy model.
- ***Operations Management*** – responsible for deploying, managing and monitoring business applications.
- ***Corporate Developer*** – responsible for the development and deployment of business applications and business components. These developers tend to be less skilled in the details of the software platform, and rightly so, their job is to deploy business value through these applications, not know the latest JDK version.
- ***Data/Information Architect*** – responsible for the development of data models and a stakeholder in interchange models for information integration.

The following table reflects the roles (Author, Secondary author, Reviewer) that these actors play in the development of the models defined in the framework above. This table is organized such that those stakeholders requiring a more abstract/less detailed view are at the top and those requiring less abstraction and/or more detail are at the bottom. We have also highlighted the columns representing the models relevant to this submission.

	Information	*Organization*	*Resource*	*Process*	*Automation*	*Strategy*	*Governance*	*Simulation*	*Financial*	*Observation*
Executives		R		R		R	R		R	
End User		R		R				R		
Business Analyst	A	A	R	A		A	A	R	A	
Process Specialist	S	S	A	S				A	R	A
Solution Architect	R		S	R	A			R		R
Corporate Developer										
Operations Management										
Data/Information Architect	R									

It is our intent to propose a set of views, their contents and a mapping to the MDA framework here. This will be provided in a later revision.

Proposed Conformance Criteria

At this time no conformance criteria have been identified.

Proposed Normative References

[1] Unified Modeling Language: Infrastructure version 2.0 (ptc/03-09-15), January 2003, OMG

[2] Unified Modeling Language: Superstructure version 2.0 (ptc/03-08-02), April 2003, OMG

[3] Business Process Modeling Notation Working Draft (0.9), BPMI.org

[4] Business Process Modeling Notation Working Draft (1.0), August 25th 2003, BPMI.org

[5] BPEL4WS 1.1 http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[6] UML Profile for Enterprise Distributed Object Computing Specification (ptc/02-02-05), February 2002

[7] Rational UML Profile for Business Modeling

[8] Unified Modeling Language: OCL 2.0 (ptc/03-08-08), August 2003

[9] Architecture of Business Modeling (br/03-11-01), OMG

Proposed List of Terms and Definitions

Automated Process

A Process that is automated, at least in some part, by a software application or process runtime (such as the WebSphere Process Choreographer). An automated process by definition is an executable process.

Business Document

A Business Document models exactly what you would expect – a paper document. As such it has no identity (in the Object sense), no behavior and no state.

Choreography

In implementing a value chain processes communicate between each other. These inter-process communications have to be governed with a contract that defines the set of messages and the sequencing of messages. This contract is known as the choreography, or global process.

Executable Process

A Process that has executable semantics, though this need not be automated as it may be executed solely by people within an organization.

Managed Process

The processes described as part of a choreography define, in effect, an interface (sometimes termed an abstract process) which has to be implemented by some actual process, these implemented processes are the managed processes. A managed process by definition is an executable process.

Partner Process

The processes described as part of a choreography are often termed partner processes.

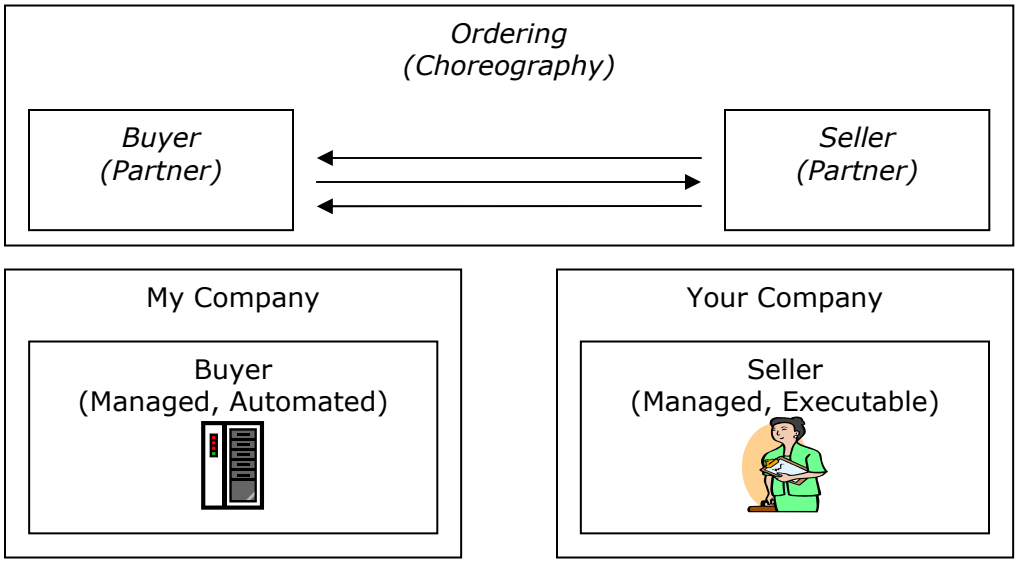
Resource

TBD

Role

TBD

The following diagram demonstrates the different process notions; the abstract processes Buyer and Seller are partners in a choreography called Ordering. Two companies agree to implement the choreography and so two managed processes are required.



My Company uses a software automation package to automate the Buyer process whereas Your Company simply executes the Seller process with an efficient sales force of trained staff.

Proposed List of Symbols

Currently there are no symbols defined as part of this submission. The appendix mapping BPMN to BPD does however introduce an alternate notation.

Business Process Definition Metamodel

The adopted UML 2.0 superstructure metamodel provides much of what is required for the definition of business processes. The following sections identify the subset of UML 2.0 appropriate for business process definition.

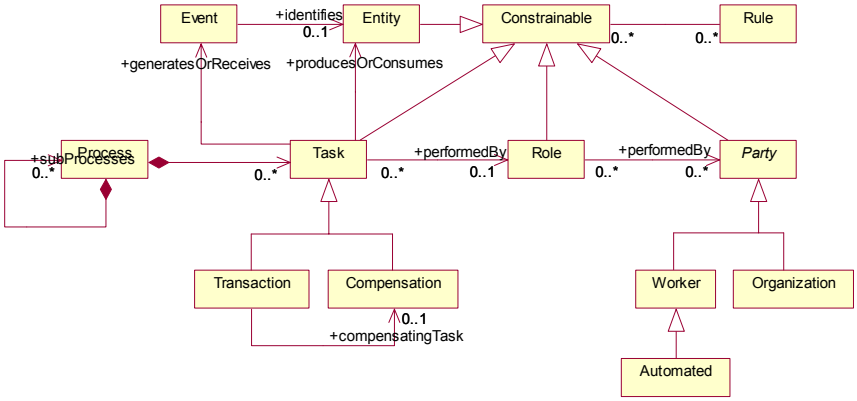
The model deals with descriptions of business processes at a level abstract from any implementation technology, even where that implementation technology may be humans executing a purely manual process. This does not imply that the model cannot and does not describe executable processes, simply that it makes no assumptions about the form of the execution. It is certainly true that current notations such as IDEF0 make no assumptions about the "implementation" of a process, and neither does the most commonly used notation, boxes-and-lines. In our case we simply intend to specify a common subset of the standard UML 2.0 superstructure that can be used to support this level of modeling.

It is worth noting that although the subset of the UML we have identified would provide the user a level of completeness that would allow them to specify and execute the complex business models there is no specific requirement that they do so. It is not usually the case that models specified at this level of abstraction are complete, it is not a requirement that they be complete.

Conceptual Metamodel Overview

This submission has identified a core set of concepts shared among a number of tools and methodologies PIM level and that represent what we believe to be an agreeable set of terms among the identified audience. We have taken this set of concepts and mapped them to UML 2.0 and developed a profile for UML 2.0 that can be used to model business process definitions well. This profile does identify a required subset of the UML 2.0 metamodel, but does not explicitly remove anything from the metamodel, thus leaving the full expressiveness of the UML open to users, tool developers and methodologists. We have also not added new metamodel level elements and so the end user does not have to learn a "new UML" and tool vendors should be able to implement the profile without major re-engineering efforts.

The following is a conceptual model of the concerns addressed by this specification. It acts as an overview and a framework which the detailed specification completes.



The concepts themselves are relatively simple; we start with a process itself, it is a container for state and behavior and has an externally visible interface, a set of operations and receptions that it responds to. Once these behaviors are invoked the process may contain sub-processes (for partitioning model behavior and reusing process classes) and tasks. Tasks may invoke behavior on other tasks and sub-processes in the processes or invoke behavior or send events that are received by processes outside the scope of their own parent process. Tasks may also identify themselves as having transactional semantics, i.e. an atomic completion of all enclosed behavior; to support this we also provide compensation activities that can explicitly model the behavior required to undo a completed transaction in the event of failure in some other part of the process.

In our view the notion of a role is a functional need of a task to be fulfilled by a resource. An organization can be responsible for the completion of a task by providing a resource owned by the organization to fulfilling the resource need of the task. Obviously tasks are carried out by people or automated systems. Frequently, these are grouped within organizational structures, such as companies, departments, or project teams. To model this we have a Resource model capturing the human and non-human resources in a business, their qualifications (expressed as the roles they can fulfill), their availability (for example, working hours), and cost structure. An Organization model serves to group resources, reflecting their employment or ownership. Resource requirements of process activities are often modeled as roles for which an activity requires resources in order to execute. Alternatively, resource requirements may be modeled directly (requesting a specific resource instance) or by specifying a resource type (any instance of this type will do).

Partitions may be used to declare resource requirements, or organizations providing the necessary resources (human and/or non-). Therefore, when a partition represents a role, this means that this role must be played by some resource in order for each task in the partition to execute; when it represents a resource (instance), or a resource type, this means that a specific resource, or any resource of the given type is required; when it represents an organization, this means that this organization is responsible for the tasks within the partition.

Note, we expect responses to recent RFPs for a complete organization model to supersede this subset of our model.

The UML also provides for the modeling of data flows within processes by using ObjectFlows within and between the activities, we simply require that these object flows are movements of business entities or business objects, rather than arbitrary data. The UML also allows us to model the sending and receiving of events, thus allowing us to decouple processes from each other and also to respond to internal events in an easy to understand manner.

Finally, many business systems employ business rules technologies to allow them to deploy systems that can be responsive by parameterizing business logic so that changes in environment or policy do not require application or process rewrites but can be accomplished through the changing of rules. We provide a description of the places where such rules can be expected, existing RFPs are addressing the need to model business rules.

Identified Subset of the UML

The following areas of the UML 2.0 Superstructure are required to support this profile:

- UML::Activities:: (*)
- UML::AuxiliaryConstructs::PrimitiveTypes
- UML::Classes:: (Dependencies, Interfaces, Kernel)
- UML::CommonBehaviors:: (BasicBehaviors, Communications)
- UML::CompositeStructures:: (Collaborations, InternalStructures, Ports, StructuredActivities, StructuredClasses)
- UML::StateMachines::BehaviorStateMachines

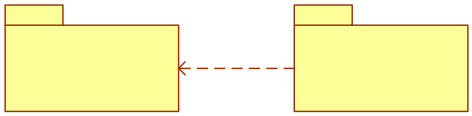
The following areas of the UML 2.0 Superstructure would be useful to support additional modeling useful to users of this profile. For example many users would expect to be able to use UseCases to capture and document information about business processes.

- UML::Actions:: (*)
- UML::AuxiliaryConstructs::Templates
- UML::CommonBehaviors::Time
- UML::StateMachines::ProtocolStateMachines
- UML::UseCases

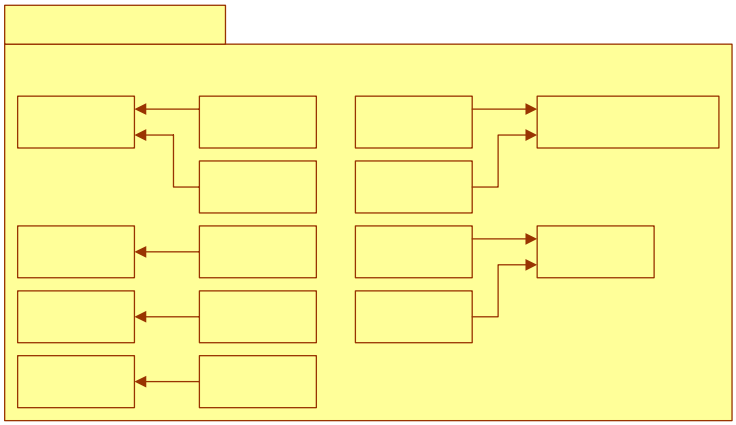
Semantics

The following sections outline these concepts and how they are realized in terms of a UML 2.0 profile. Again, it is important to stress that an objective of the initial submission team was to not require an extension to the UML (if this was feasible – it is not a firm requirement) and as such a we present here a consistent subset of the UML where the elements of this profile are only used to disambiguate certain model elements.

Note that many of the concepts do not have explicit profile elements; that the existing UML 2.0 metamodel provides the right semantics and the structure of the profile allows their use in an unambiguous manner. After this table we will list the packages of the UML 2.0 superstructure required to support this profile and the additional modeling capabilities we expect our users to want.



The following diagram shows the current state of the profile, note this is the profile notation for UML 2.0.



Note, this is currently incomplete and requires a complete Resource model including access control, correlation etc.

Process Stereotype

Extends

CommonBehaviors::Communications::Class

Semantics

Represents the external view of a process, denotes its contractual interface. Also acts as a container for private state and behavior. State date may be modeled using properties on the Class and the behavior modeled as a Process Flow (see below).

The presence of Operations and/or receptions on a «Process» class as well as a Process Flow is currently being discussed.

The Class from CommonBehaviors::Communications merges BehavioredClassifier from BasicBehaviors and Class from Kernel; BehavioredClassifier and Kernel::Class both inherit from Kernel::Classifier.

PROPERTIES

None.

WELL-FORMEDNESS RULES

None.

Process Flow Concept

USES

Activities::BasicActivities::Activity

SEMANTICS

A process has an associated classifierBehavior implemented by an activity. In the UML a BehavioredClassifier may have an associated behavior which in this case represents the process behavior.

There is no restriction on the contents of an Activity representing a Process Flow.

WELL-FORMEDNESS RULES

The behavior associated with a «Process» Class is assumed to represent the process flow.

Shared Sub-process Concept

USES

CommonBehaviors::Communications::Class

SEMANTICS

Sub-processes are defined with nested «process» Classes.

We separate out the notion of sub-process, in some cases is a sub-process may be shared between different processes, or at least invoked from multiple locations within an enclosing parent. There are also sub-processes that are not shared, not invoked from multiple locations but are created to manage the size or complexity of a process or to denote some logical grouping of actions.

WELL-FORMEDNESS RULES

None.

Embedded Sub-process Concept

USES

Activities::StructuredActivities::StructuredActivityNode

SEMANTICS

These specify fine-grained sub-processes (see above).

WELL-FORMEDNESS RULES

None.

Atomic Task Concept

USES

Activites::BasicActivities::Action

SEMANTICS

Represent atomic steps in a process. A step may comprise a value calculation, update or read of state data, sending events or calling of sub-processes; there are no restrictions on the kinds of actions from UML allowed.

Note that to sequence between two atomic tasks and a shared sub-process a CallBehavior action is required.

WELL-FORMEDNESS RULES

None.

Role/Participant Concept

USES

Activities::IntermediateActivities::ActivityPartition

SEMANTICS

Any partition on an activity diagram within a process class may be used to describe a role played by a party or responsibility of an organization in a process. As such the "*represents"* meta relationship must identify a class stereotyped either *Worker* or *Organization*.

WELL-FORMEDNESS RULES

None.

Data Object Concept

USES

Activities::BasicActivities::ObjectNode

SEMANTICS

Represents the flow of data input, out of or between activities. As these data objects are passed between tasks which may also involve the crossing of application, organization or enterprise boundaries it is not permissible to pass an actual instance of an Entity (representing an application view of some business data) but a Document representing some understood view of the business data. For more details see «Entity» below.

WELL-FORMEDNESS RULES

All Data Objects should be of a type that is a «Document».

Sequence Flow Concept

USES

Activities::BasicActivities::ControlFlow

SEMANTICS

Identifies the sequencing of activities, sequence flow and data flow are used in conjunction in a model however tools may wish to simplify this into a single connector concept.

WELL-FORMEDNESS RULES

None.

Data Flow Concepte

USES

Activities::BasicActivities::ObjectFlow

SEMANTICS

Identifies the flow of Data Objects between activities. In some business modeling tools a single flow connector is used which may or may not carry data.

WELL-FORMEDNESS RULES

None.

Exception Flow Concept

Uses

Activities::ExtraStructuredActivities::ExceptionHandler

Semantics

Identifies the flow of control from one activity to an exception handler in the case of an error.

Well-formedness Rules

None.

Transaction Stereotype

Extends

Activities::CompleteStructuredActivities::StructuredActivityNode

Semantics

Identifies a task as having transactional semantics; all actions enclosed in the structured activity participate in the transaction context. The semantics of a transaction here are not those of a typical database-style transaction. There is no 2-phase commit; there is no automatic rollback management. If, at some later point it is deemed necessary to undo a transaction then an explicit compensation task is performed.

Properties

None.

Well-formedness Rules

A transactional task may only have zero or one flows to a compensation activity.

Compensate Stereotype

Extends

Activities::CompleteStructuredActivities::StructuredActivityNode

Semantics

Identifies the task that is invoked when a transactional task has to be undone.

PROPERTIES

None.

WELL-FORMEDNESS RULES

None.

Entity Stereotype

EXTENDS

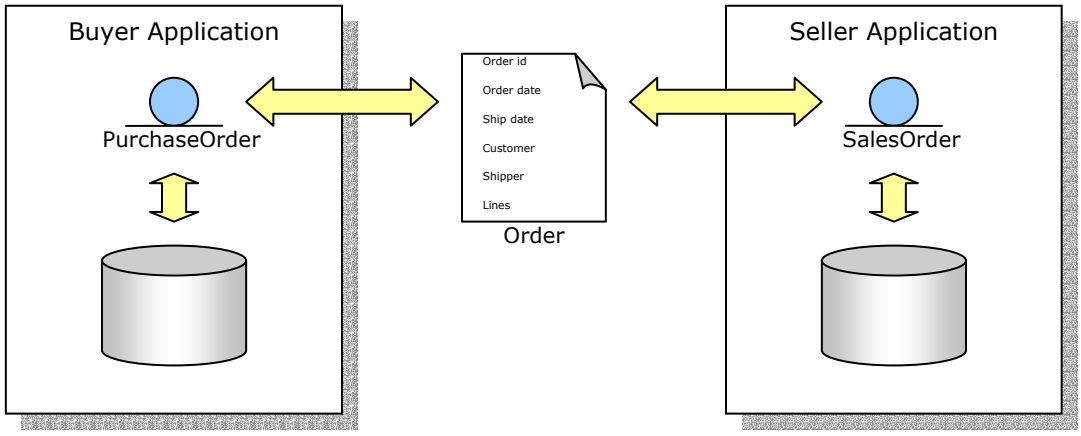
Kernel::Class

SEMANTICS

An entity acts as a container for a set of business data.

The use of Class as our meta-class implies that the entire information model does not have to consist of stereotyped classes. It is not required that all classes representing the static structure of an Entity be stereotyped as «Entity». This is an implementation decision.

The relationship between Entity and Document needs to be described. A business entity has an object-oriented flavor, it has identity, behavior and a state-space and so can be a hard concept to grasp from a business user perspective. In business it is common that artifacts are described as documents, they have no implicit identity; they have no behavior of their own and no state-space. Documents are related to Entities however, they represent views of an Entity, snapshots if you will. The following diagram shows how an "Order" Document is related to two Entities in this case, the "PurchaseOrder" the buyer generated and the "SalesOrder" the seller fullfils.



PROPERTIES

None.

WELL-FORMEDNESS RULES

None.

Document Stereotype

EXTENDS

Kernel::DataType

SEMANTICS

Acts as a view onto an entity, the primary artifact used to pass information between tasks in a process. If you consider an «Entity» that has a state model associated with it (physically in the model, or simply conceptually) then one would consider generating a «Document» for each state. For example, our Purchase Order «Entity» May have the following states:

- Initial
- Verified
- Accepted
- Shipped
- Fulfilled
- Completed

However, at each state only certain properties/fields are applicable (for example it is not possible to add a meaningful value to the "shipped-on date" when in the "initial" state) and so the corresponding «Document» need not include this property at all.

PROPERTIES

None.

WELL-FORMEDNESS RULES

Each «Document» should have a dependency to an «Entity» from which it is derived.

Identity Stereotype

EXTENDS

Kernel::Property

SEMANTICS

This is simply a stereotype that allows the modeler to identify a property on an entity as a potentially identifying one. There are no restrictions on the kinds of properties to which this stereotype may be applied, and no restriction on the number of identifying properties on an entity. However when more than one property is marked as an identity it is undefined the order in which the identity is assumed to be calculated.

PROPERTIES

None.

WELL-FORMEDNESS RULES

May only be applied properties of «Entity» Classes.

Event Stereotype

EXTENDS

CommonBehaviors::Communications::Signal

SEMANTICS

A business event is used as a primary technique for decoupling processes through sending and receiving events. This does not imply that this is the only mechanism, one can directly call behavior on a partner process, in which case either the actual process is known or some interface/operation is known in the target. In the case of the sending of an event (particularly broadcast) less information is needed about the target.

Note that we do not explicitly stereotyped the sending receiving of events here, it is well covered by the UML 2.0 activity metamodel.

PROPERTIES

None.

WELL-FORMEDNESS RULES

None.

Rule Stereotype

EXTENDS

Constraint

SEMANTICS

Business rules govern the behavior of elements in a process, specifically the activities, participants and entities.

We expect to replace this "hook" with the outcome of the Business Rules RFP.

PROPERTIES

None.

WELL-FORMEDNESS RULES

None.

Resource Concept

SEMANTICS

This section of the model is currently incomplete.

Worker Stereotype

EXTENDS

Communications::Class

SEMANTICS

A worker is an individual contributor to the organization and may play roles in processes. A Worker class may not contain other classes. A worker has a property to identify whether they carry out their tasks in a manual or automated fashion.

The use of Class here is problematic when describing a human worker; Actor is another proposal however this (as a subclass of kernel::Classifier) does not have behavior – a prerequisite when describing autonomous agents.

The relation between this and the resource model is currently undefined.

PROPERTIES

None.

WELL-FORMEDNESS RULES

None.

Organization Stereotype

EXTENDS

Communications::Class

SEMANTICS

The use of Class for Organization is similarly problematic; we look to the organizational structure RFP to solve this.

The relation between this and the resource model is currently undefined.

PROPERTIES

None.

WELL-FORMEDNESS RULES

None.

Model Management Concept

USES

Package

SEMANTICS

As models grow it is important to not only decompose processes using process classes but also to be able to separate, store, share and version model fragments. UML Packages are provided and recommended for this.

PROPERTIES

None.

WELL-FORMEDNESS RULES

None.

Comment: Need to flesh out the following areas:

1. Resources, including resource allocation policies and so forth.
2. Manual vs. Automated actions.
3. Connection from Business Rule to the BR Submission.
4. Addition of vocabulary metamodel for element classification.
5. Resolve the issues (from 12/18 call) around the semantic definition of entity.

Examples & Discussion

This section will demonstrate the profile above using examples that highlight the key concepts and also some of the variability provided by the underlying UML metamodel.

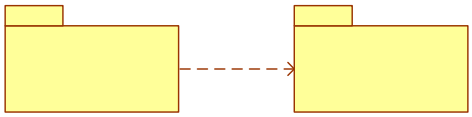
Our example process is an Order Management Process[3], containing sub processes for:

- **Quote and Order Entry**; Allows partners to exchange price and availability information, quotes, purchase orders and order status, and enables partners to send requested orders, or shopping carts, to other partners.
- **Transportation and Distribution**; Enables communication of shipping- and delivery-related information with the ability to make changes and handle exceptions and claims.
- **Returns and Finance**; Provides for issuance of billing, payment and reconciliation of debits, credits and invoices between partners as well as supports product return and its financial impact.
- **Product Configuration**; Enables development of custom configurations to feed the order management process and supports the effort for engineering changes by providing supply-chain communications for implementation.

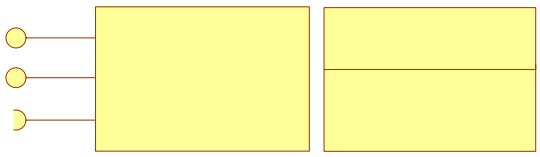
Process Representation

Comment: Need to explicitly describe and show, through example, the notion of a collaborative process. The Borland/Data Access/EDS/88 Solutions team firmly believe that collaborative processes are a separate and first-class concept. We need to describe our view and contrast it with a description and example to be provided by Cory.

Following is the UML 2.0 notation for applying a profile to a package, in this case the package containing the definition of our process(es).



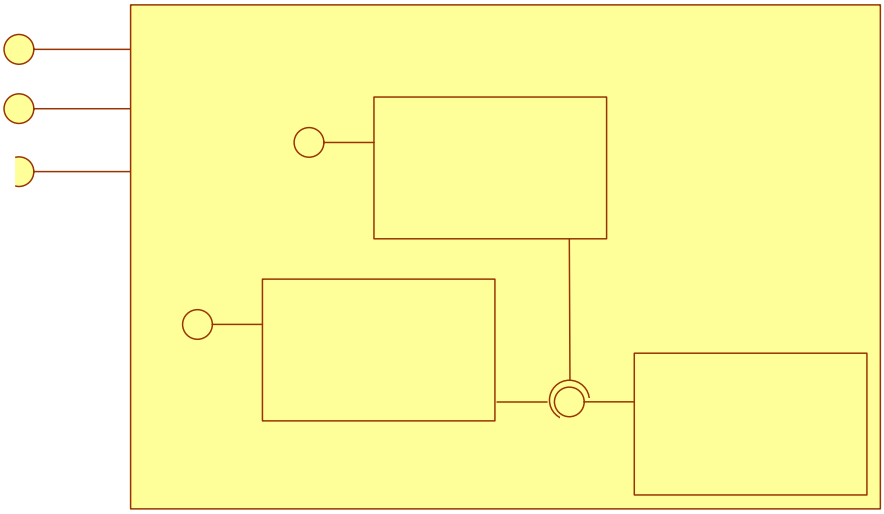
The following is the expected representation of a «Process», using UML 2.0 notation; now this does look daunting, which is why we expect to provide a business-level notation. We see a UML 2.0 Class, stereotyped as «process» with both provided and required interfaces. These interfaces describe the messages and events that the process responds to and, in the case of the required interfaces, describes the messages and events that clients must respond to.



Sub-Processes

To define a sub-process one simply has to create a class nested within the «process» above. We then connect the interfaces on our process to the interfaces on the sub-processes that implement them.

[3] Based on RosettaNet Cluster 3 Order Management PIPs.



This leaves open a question over how an activity (behavior) is associated with a process class. The UML 2.0 allows for a number of ways, the two interesting are as follows:

- An Activity (As a subtype of Behavior) can be directly associated with a BehavioredClassifier (of which Class is a subtype) via the classifierBehavior metarelationship. This then models the behavior of the process.
- A Class has implementations for each of the operations and receptions exposed on its interfaces, these may have behaviors which only model the particular behavior of that operation.

QuoteManagement

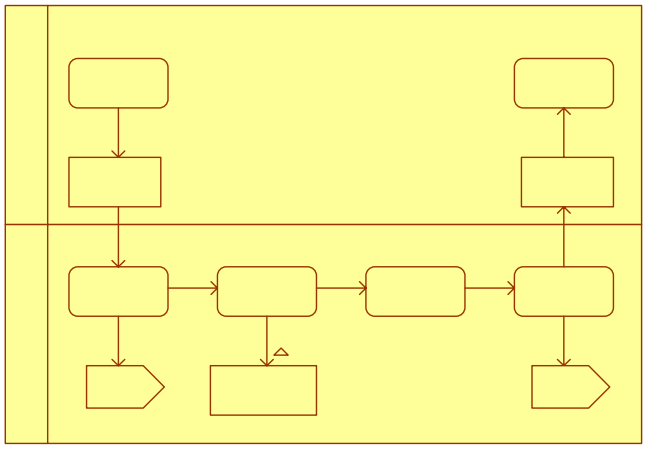
OrderManagement

So, for example when OrderManagement requires to validate an order it may signal to the OrderValidation behavior or invoke the behavior of a particular operation to address this need.

OrderManagementCallbacks

Flows

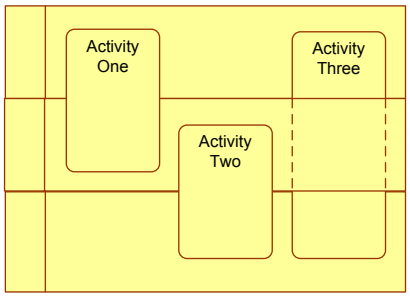
Following is an example of a flow describing the behavior of an activity (Order Management).



You can see in the example above the use of actions, object flows, raised signals (SendSignalAction) and exceptions (RaiseExceptionEventAction).

Partitioning

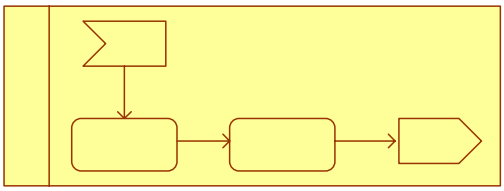
It should be noted on the diagram above that the partitions shown are being used to represent which tasks are executed by the "Buyer" and "Seller" play in the process, however partitions can also be used to show other concerns, such as responsibility, location etc. Therefore, we provide for roles being represented as partitions, but do not restrict partitions to only represent roles. Another problem with roles is demonstrated in the example below.



It is common for an activity to be a collaboration between roles, so naturally we can place the activity in more than one partition; so for example, activity one is performed by partition A and B, activity two by partition B and C. But now, what about activity three? It is performed by partition A and C, we can't draw that, so … ?

Events

In the example above we see that the process generates two events, the order acknowledgement and the new order event. It is this new order event that becomes particularly interesting as there are many other processes interested in the fact that an order has been accepted. For example, forecasting sales requires continual assessment of current order trends, so our forecasting process may model itself as below.



This shows that the Forecasting process has a signal reception, waiting for the signal to arrive, when it does it can process the order, update its forecast and then using another event tell the business that a new forecast is available.

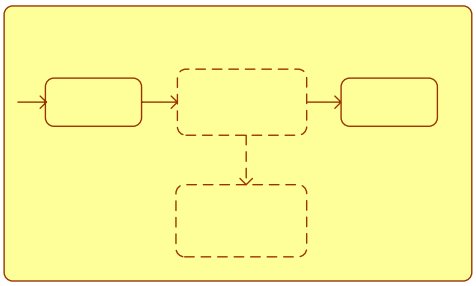
This decoupling of processes through events is extremely important in reducing the dependency of one process on another and allows for a process to publish the events it raises and another process to explicitly publish the receptions that it supports.

Note, at a metamodel level this requires the use of BroadcastSignalAction to support the decoupled reception, point-to-point signaling uses SendSignalAction.

Discuss this approach in respect to directly coupling using CallBehavior and CallOperation Actions.

Transactions

Transactions are an important part of business modeling, to denote a set of actions that have transactional behavior, whether they be implemented in software or simply as a group of collaborating human beings. The following diagram shows how an action can be stereotyped as a transaction and how we can also denote the compensation action that is executed when the transaction fails.

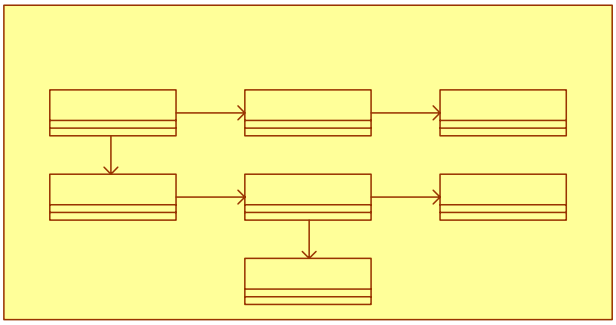


Information Models

We introduce a stereotype, «entity» to act as the traditional "Business Object" which represents the canonical and complete notion of a business concept (such as Purchase Order, Customer etc.). This revision also introduces the model element «Document» that corresponds to the common notion of a "Business Document"; it is the actual level of information passed around within a process. For example, an order has a header and order lines, processes do not move individual lines around they move the order between actions.

The following diagram shows an «Entity» Order and two «Documents» Accepted_Order and Completed_Order that are the actual data passed between tasks in our process.

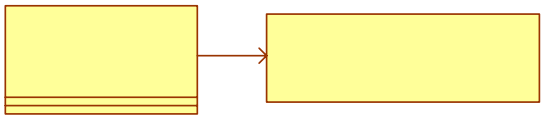
An Order is then modeled showing its internal structure with a combination of internal, private classes and shared, public classes. In the example below you could see that Contact, Address and Product are common classes in a reusable package whereas Header, Line and Item are specific to the implementation of the Order and so are nested within the class itself.



Another stereotype introduced into the information model is the notion of identity, a (one or more) property of a class that acts as its identifier. This is not the same as object identity for a programming language or a primary key in a database table, it is the logical identity and is therefore implementation independent.

Above we saw how events are sent and received within a process; we now need to understand how these are described. In general business events are produced in response to, or communicate the, change in state of a business entity and some business events are generated according to set time.

The diagram below shows a business event for denoting a new order actually associates to the Order that has been accepted.



Organization

People interact with and implement processes in one of three ways:

- A process may only exist as a collaboration between people with no automation at all.
- People may perform roles in a process, interacting with automated elements, such as filling in online forms.

People may simply act as exception handlers…

Part III – Changes to Existing Specifications

At this time no changes are required to existing specifications.

Part IV - Appendices

Appendix A: A UML Profile for BPD with a Mapping to BPEL4WS

This appendix introduces a Unified Modeling Language (UML) profile for automated business processes (business process definition PSM) with a mapping to the Business Process Execution Language for Web Services (BPEL4WS or BPEL). The concepts supported by the profile are based on those supported by BPEL. This approach enables automated business processes to be modeled using UML tools such as Rational Rose and Rational XDE and then translated into BPEL documents that can be executed on a BPEL runtime.

The profile described in this appendix is made available as a draft for comment and should be considered as a work in progress.

Model-driven business integration

Graphical modeling is a popular way of drawing designs, showing business information, communicating views and perspectives on IT systems, and defining executable things such as programs, flows, simulations, and rules. UML is a widely-used industry standard for graphical modeling, but there are many other representations and tools.

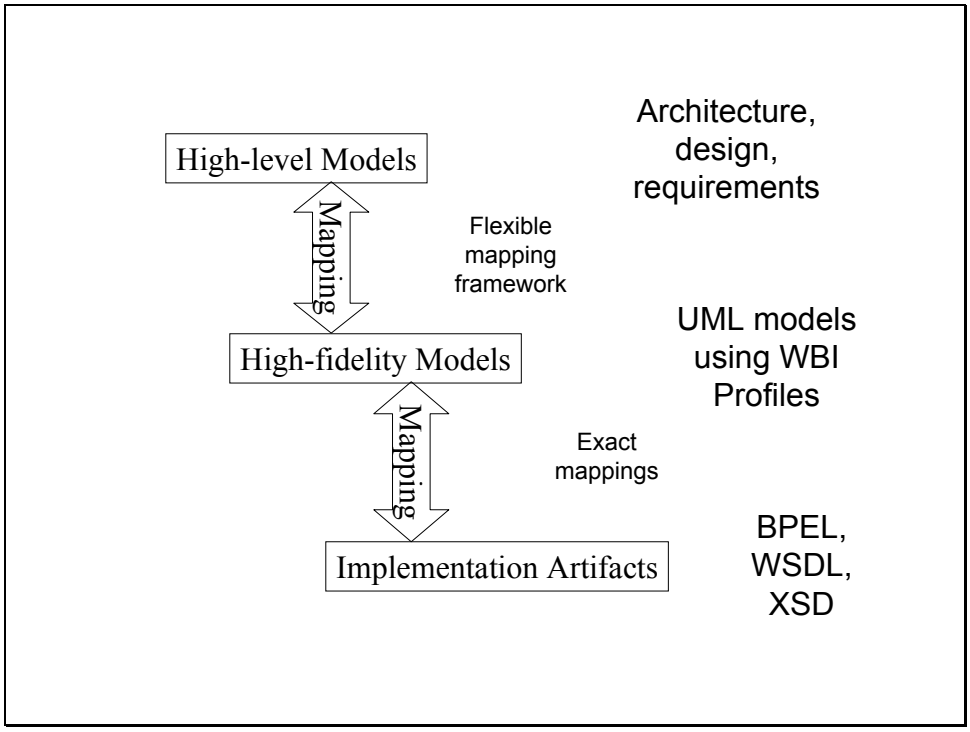


Figure 1 Some kinds of model important to MDBI

Figure 1 shows a distinction between models of different kinds.

High-level models

A *high-level model* does not represent the details of implementation artifacts. A high-level model shows only those aspects that are needed for another purpose, such as representing design, architecture, or requirements. It can show items that are not implemented or that can be defined in detail later; it can include representations of legacy systems, where only the interfaces need be specified.

High-fidelity models

A *high-fidelity model* is capable of representing the details of implementation artifacts, so that someone using a drawing tool does not need to see the textual representation of the model. (A person working with the model can see text, such as names and annotations, that is included in the drawing.) WebSphere Studio

provides tools for such high-fidelity models for runtime products such as WebSphere MQ Workflow and WebSphere MQ Integrator.

Implementation artifacts

Implementation artifacts are sometimes seen as kinds of models. Implementation artifacts in XML can often be viewed with convenient tools, but they are textual, not graphical.

In model-driven business integration, UML is used to exploit its status as a standard modeling language and to ease the transformation to and from high-level models. UML is a well-known popular standard; it is frequently used to express higher-level models, and it is capable of carrying the detail of a high-fidelity model. It is desirable for UML models to be able to express all the details of implementation artifacts, but the goal is to allow modelers the flexibility to express less than complete detail without prescribing how much detail is needed. Refinement is then a natural activity of adding further detail, while still conforming to an industry standard, rather than having to switch to a quite different modeling style. High-fidelity models contain sufficient detail be mapped directly to implementation artifacts.

One aspect of business integration is the development of automated business processes. The Business Process Execution Language for Web Services (BPEL4WS) provides an XML-based notation and runtime semantics for business processes that communication through web services.

This document specifies how to create high-fidelity UML models that describe business processes and which can be mapped to BPEL4WS.

Automated business processes

The term **automated business processes** is used to refer to loosely-coupled interacting software components that implement potentially long-running business-relevant tasks. Examples of automated business processes include purchase order, travel booking, and auction processes.

BPEL 1.0 as a platform for executing automated business processes

The Business Process Execution Language (BPEL) provides an XML notation for describing automated business processes and provides semantics for executing business processes.

From the BPEL 1.0 specification:

"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.

BPEL4WS provides a language for the formal specification of business processes and business interaction protocols. By doing so, it extends the Web services interaction model and enables it to support business transactions. BPEL4WS defines an interoperable

integration model that should facilitate the expansion of automated process integration in both the intracorporate and the business-to-business spaces."

The current version of this profile is concerned with the modeling of executable business processes.

The BPEL specification is available at the following location:

http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

BPEL 1.1

A BPEL 1.1 specification has recently been made available. The extensions supported in BPEL 1.1 will be supported in a future version of this profile.

UML profile

A *UML profile* is both a subset and extension of UML. It allows UML to be customized for modeling in a particular context. Customization is achieved by defining the following items:

- The subset of model elements to be used in the profile
- Stereotypes, tagged values and constraints to be used with specific model elements

The UML profile for automated business processes described in this document is a profile of UML 1.5.

Tools vary in the support they provide for profiles. Most tools support stereotypes, some support tagged values. Few tools support constraints and subsetting; that is, they do not enforce constraints and they do not prevent the modeler from using elements that are not in the profile.

The profile described in this document uses a subset of the UML modeling elements, and introduces some stereotypes to customize them. It does not use tagged values. Constraints are described informally, where appropriate. In addition, some tips are provided on how to use existing CASE tools, specifically Rational XDE and Rose, for creating models using the profile.

Defining a business process

An initial example – The purchase order process

This section introduces the UML profile through an initial example that defines a simple purchase order process. This example is taken from the BPEL 1.0 specification.

"On receiving the purchase order from a customer, the process initiates three tasks in parallel: calculating the final price for the order, selecting a shipper, and scheduling the production and shipment for the order. While some of the processing can proceed in parallel, there are control and data dependencies between the three tasks. In particular, the shipping price is required to finalize the price calculation, and the shipping date is required for the complete fulfillment schedule. When the three tasks are completed, invoice processing can proceed and the invoice is sent to the customer."

The following diagram (a UML activity graph) provides an overview of the process. The icons in the lower left-hand corner of each activity indicate that the details of the activity have been hidden; this detail is introduced later in this section.

In summary, the diagram shows that the process has four ports: customer, invoiceProvider, shippingProvider and schedulingProvider, which are represented by the rectangular partitions. Activities that involve a message send or receive operation through a port appear in the corresponding partition. The arrows indicate the order in which the process performs the activities.

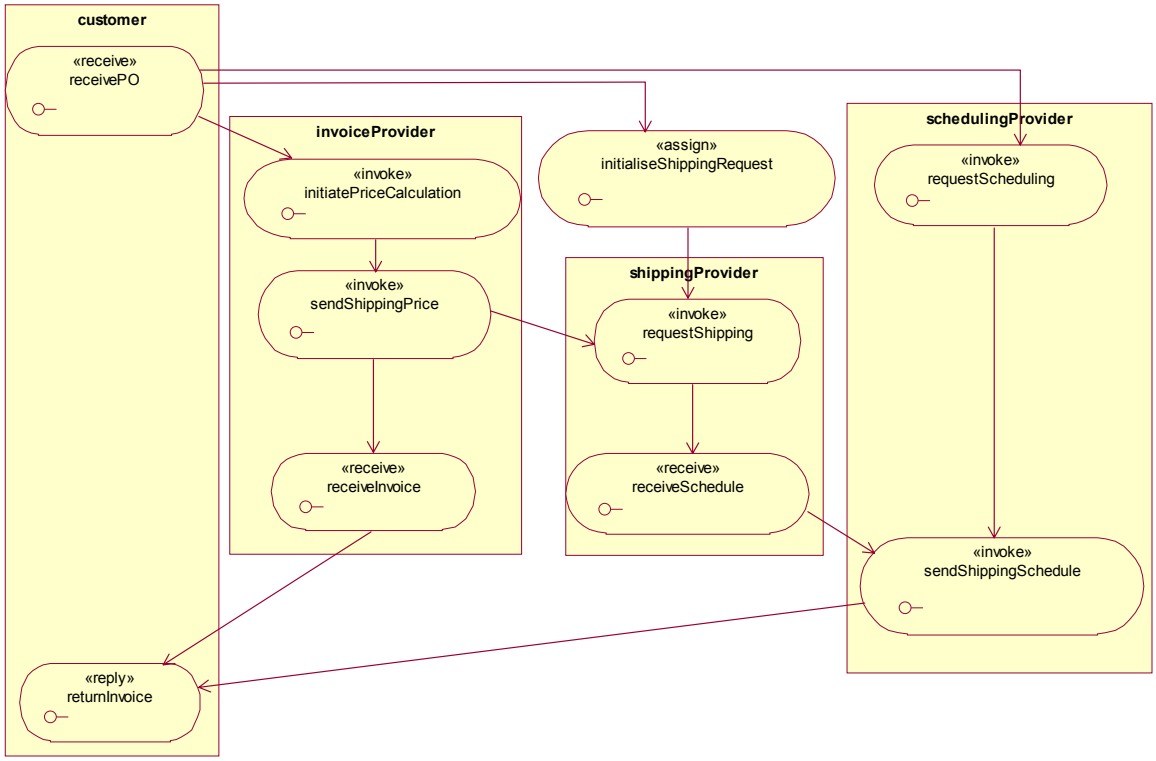


Figure 2: Purchase order process overview

1. The purchase order process begins by receiving a purchase order request from a customer (through the customer port).
2. The initiatePriceCalculation, initialiseShippingRequest and requestProductionScheduling activities begin executing, triggering further activities as they complete. Note that the requestShipping activity requires that both the initialiseShippingRequest and sendShippingPrice activities have taken place before it begins.
3. The returnInvoice activity returns a response back to the customer.

Figure 2 is only one of the diagrams required to model this process completely. Detail is elaborated using further diagrams supported by the profile.

The process participates in a protocol through each of its ports. A protocol links a pair of complementary roles and shows how these roles interact. Each role optionally provides a set of interfaces, which are required by the complementary role.

Data types are represented in the profile by UML classes. The class diagram in Figure 3 shows the classes that are used as operation parameter types in interfaces that are

provided or required by roles in the purchase order process protocols. In this case, most of the classes needed are already defined in an existing package, PurchaseDataTypes. The stereotype <<external>> applied to the PurchaseDataTypes package indicates that package is defined outside of the current model and should not have artifacts generated in a mapping to a target platform (such as BPEL). A new package, Purchase, is introduced. This has an import dependency on PurchaseDataTypes so that the existing types can be reused[4]. One further class, PO, is introduced. This has purchaseOrder and customerInfo parts, where the PurchaseOrder and CustomerInfo are classes defined in the PurchaseDataTypes package.

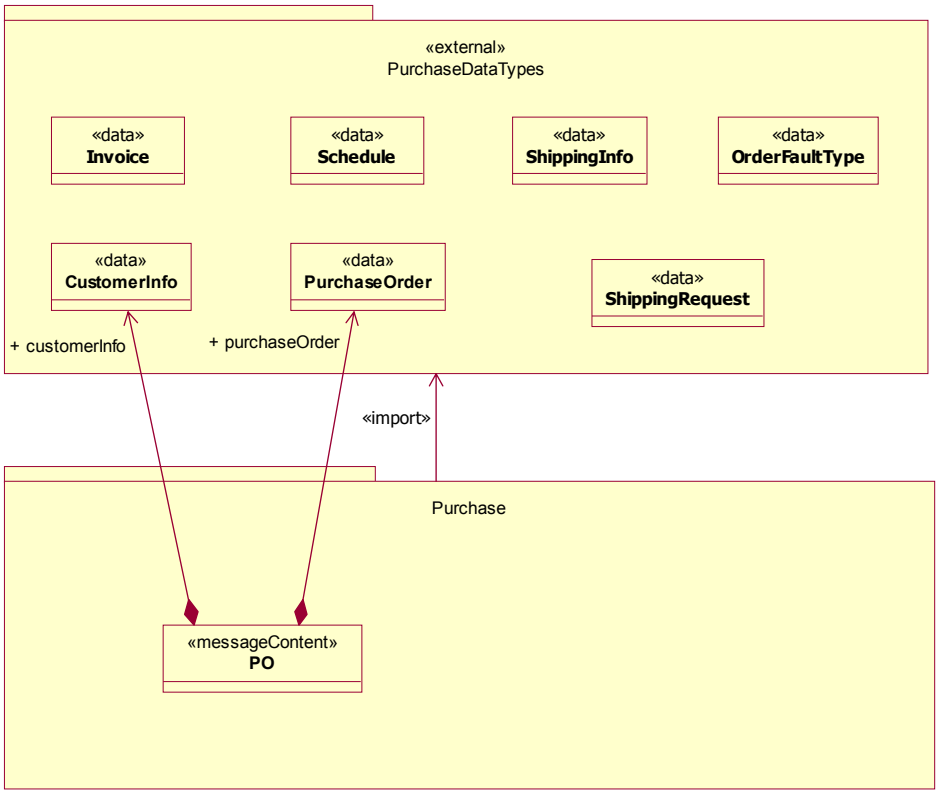


Figure 3: Message classes for use in the purchase order process.

Figure 4 introduces the interfaces that are provided or required in the purchase order protocols. These interfaces are also placed in the Purchase package.

[4] Use of an <<import>> dependency means that names from the imported package can be used without qualification.

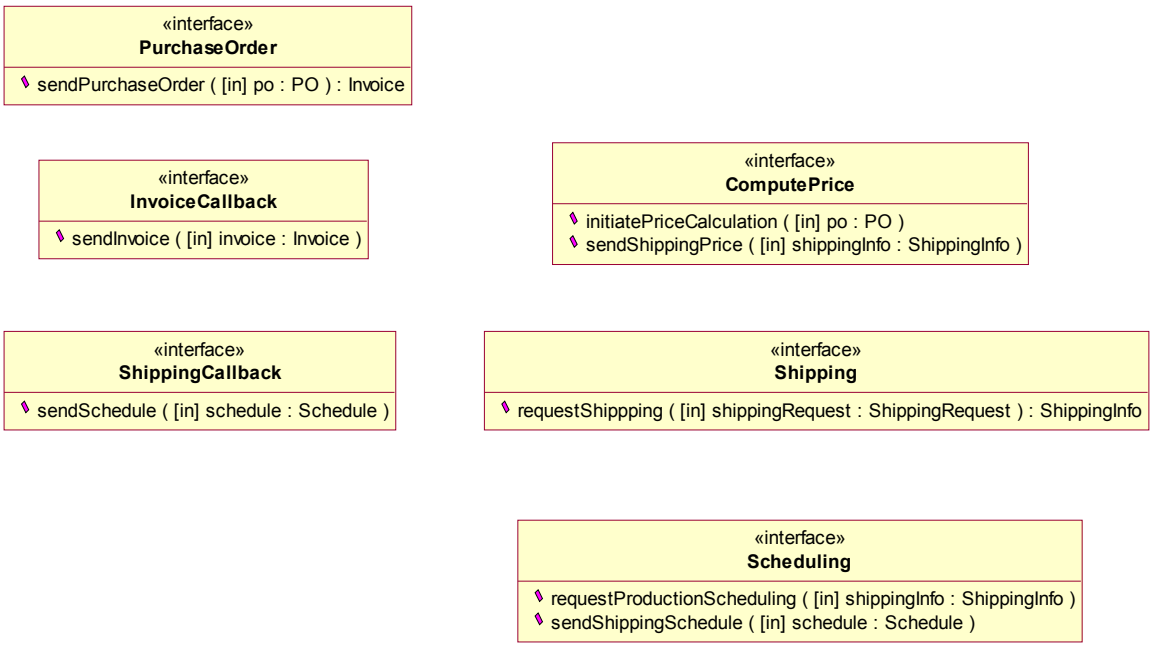


Figure 4: Interfaces uses by the purchase order process

Figure 5 introduces the protocols in which the purchase order will play roles and interact with partners playing the complementary roles. The roles played by the purchase order process appear on the left. Each protocol is placed within a subpackage of the Purchase package, the roles are placed within the corresponding protocol package. The Purchase package contains the classes and interfaces used by the protocols.

In the Purchase and Scheduling protocols only one role provides an interface; the other role describes a requester of the service. The Invoice service introduces an InvoiceService role that provides the ComputePrice interface and requires the user of that interface to provide an InvoiceCallback interface for an asynchronous response. The Shipping protocol also describes a two-way interaction.

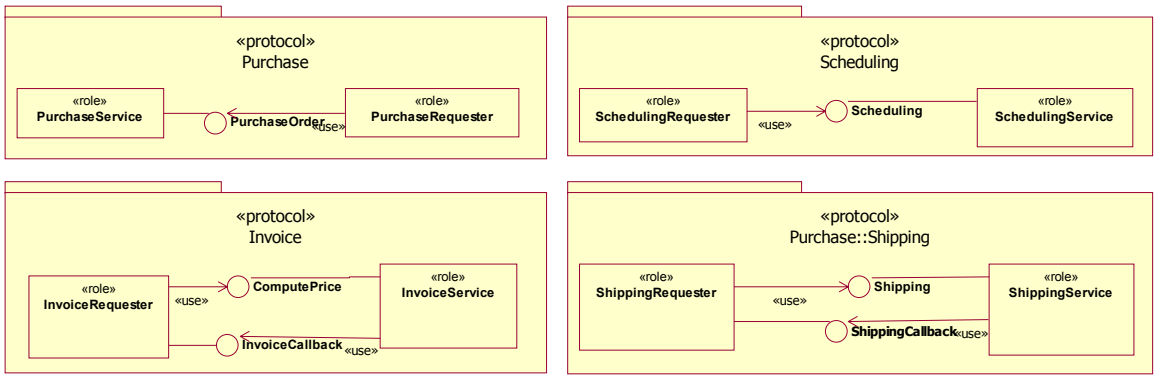


Figure 5: Protocols and roles

At this point, all the definitions that the purchase order process depends on have been provided. Although these definitions have been introduced within the purchase order example they can be reused independently of the purchase order process. If partners of the process are also modeled as automated business processes, then they reuse these

definitions. Other processes that provide some or all of the same roles as the purchase order process can also reuse the definitions.

This example only needs to show how the purchase order process fulfills its roles (through its ports). It does not need to show how the complementary roles are fulfilled by the partners of the process.

Figure 6 introduces a class called PurchaseOrder, stereotyped as <<process>>, that represents the purchase order process. The process has four ports corresponding to the four partners with which it interacts (according to the protocols introduced in Figure 5).

Each port is a connection point for a partner with which the process interacts. The interfaces that are provided or required through that port are defined by the role that the process plays in the protocol. A port is represented by an association (stereotyped <<port>>) to the role *played by the process*.[5] Recall that the protocols define the interfaces that must be provided and are required by each role.

The roles are qualified by the names of the protocol because the protocol packages are not imported directly, the Purchase package containing the protocol packages is imported by the PurchaseOrderProcess package. (The protocol packages could have been imported directly by the PurchaseOrderProcess package, but the qualification by protocol provides useful additional information to the reader of the diagram.)

The PurchaseOrder process provides the PurchaseOrder interface through its customer port, and requires the Scheduling interface through its schedulingProvider port.

The invoiceProvider port supports two-way communication: the PurchaseOrder process requires the ComputePrice interface and provides the InvoiceCallback interface, as specified by the InvoiceRequester role of the Invoice protocol. The shippingProvider port is also two-way.

The state of the process is shown as attributes of the PurchaseOrderProcess class, whose types were introduced in the Purchase package or imported from the PurchaseDataTypes package.

[5] UML 2 provides a semantics and graphical notation for ports. Because this is not available in UML 1.x, ports are represented using the constructs that are available.

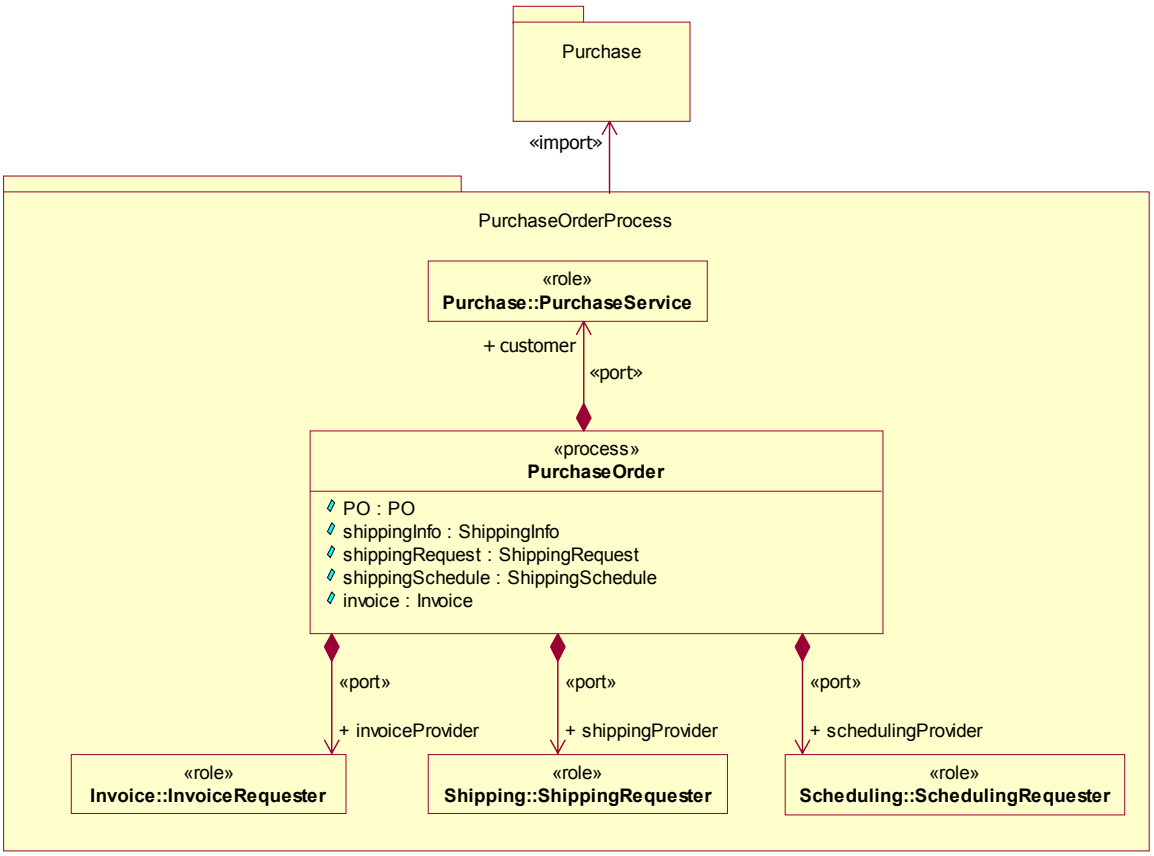


Figure 6: PurchaseOrder process ports and state

The activity graph in Figure 7 shows the behavior of the purchase order process. It is the same as Figure 2, except that more detail has been revealed. The activity graph is attached to the PurchaseOrder process class.

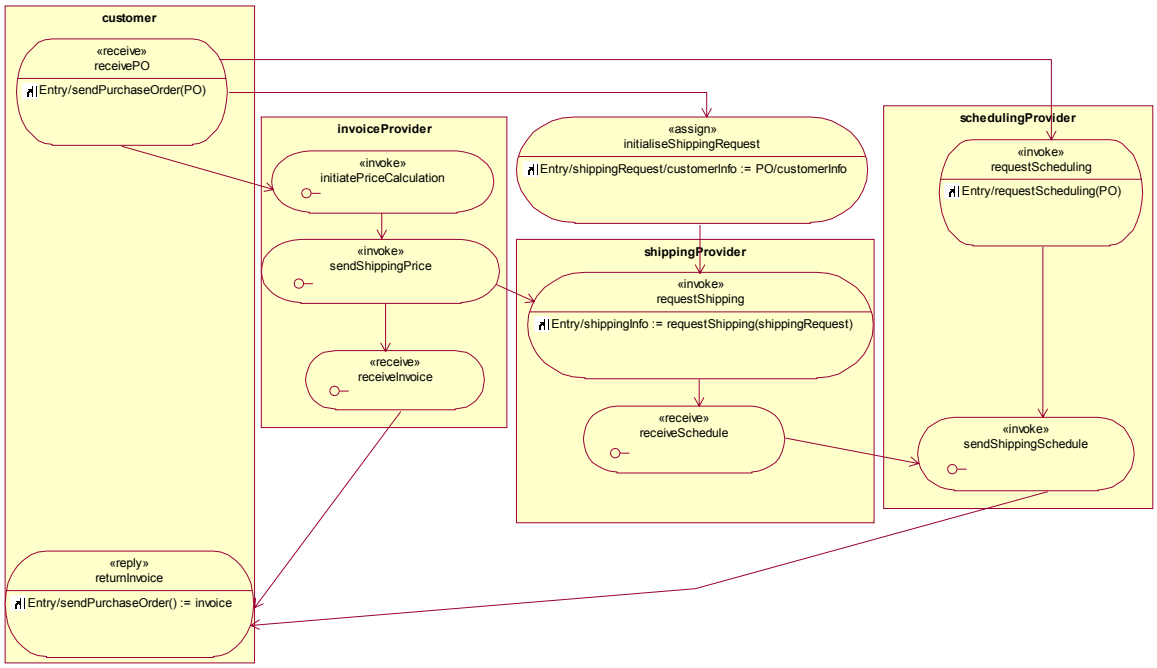


Figure 7: Activity graph for the purchase order process

A partition (*swimlane*) is introduced corresponding to each port. Activities which represent an interaction through a port (either incoming or outgoing) are placed in the appropriate partition with control flow arrows indicating the sequencing of activities.

Each activity has a descriptive name and an entry action detailing the work performed by the activity. Typically the actions would be hidden on an activity graph diagram. Figure 7 expands a number of activities to show the detail.

The receivePO activity is stereotyped with <<receive>> to indicate that it corresponds to the receipt of a message through a port, in this case the customer port. The action expression, `sendPurchaseOrder(PO)`, indicates the operation that is invoked, through the customer port, on the purchase order process and the attribute (the name of which appears as a parameter to the operation) into which the incoming message is placed. The sendPurchaseOrder operation signature is defined in the PurchaseOrder interface in Figure 4, which the PurchaseOrder process provides through its customer port.

The returnInvoice activity is stereotyped with <<reply>> to indicate that it returns a result to the invoker of an earlier corresponding <<receive>> activity. Both activities refer to the same operation, sendPurchaseOrder. The receive activity specifies the attribute into which the incoming message is placed. The reply activity specifies the attribute from which to take the reply message. The expression

```
sendPurchaseOrder() := invoice
```

indicates that the value of the invoice attribute will be returned as a response to the sendPurchaseOrder operation (which must have previously been invoked).

The requestShipping and requestScheduling activities are stereotyped as <<invoke>> to indicate that they invoke an operation through a port. The requestScheduling activity does not specify an attribute into which to place a response and is therefore one-way (asynchronous). The `requestProductionScheduling(PO)` expression indicates that the requestProductionScheduling operation is invoked (through the schedulingProvider port) with the value of the PO attribute. The requestShipping activity is two-way (synchronous), the expression

```
shippingInfo := requestShipping(shippingRequest)
```

indicates that the requestShipping operation is invoked (via the shippingProvider port) with the value of the shippingRequest attribute as a parameter; the synchronous response to this invocation is placed in the shippingInfo attribute.

The initialiseShippingRequest activity does not involve interaction with a partner and is therefore not placed in a partition. It performs a simple copy of data from one variable to another. The `:=` operator is used for assignment.

As in BPEL, data passing in this example is performed through variables scoped to the process. Those variables are represented as attributes in the class that represents the process. For example, the receivePO activity receives a value (that the customer port) into the PO variable which is then read by the requestScheduling activity and used as an input to the requestScheduling operation that it invokes (through the schedulingProvider port).

Dependency management

Concepts

Models are typically structured into packages to support namespace management and provide appropriate units for reuse. Figure 8 shows the package structure for a LoanApproval example. Dependencies between packages are represented using import dependencies.

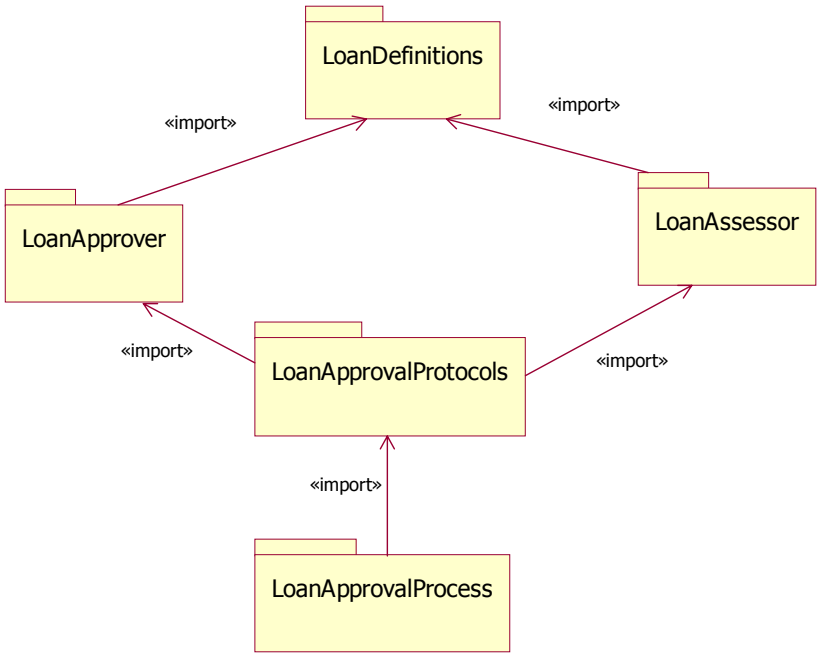


Figure 8: Example of package dependency

Notation

Packages are used to group elements and provide a namespace. In UML 1.x, dependencies between packages are referred to as *permissions*. Package dependency is indicated with a

UML <<import>> permission arrow as shown in Figure 9. Use of an import permission allows the importing package to refer to imported elements without namespace qualification. For example, if a class X is defined in Package1 then Package2 can refer to class X directly, there is no need to use Package1::X.

The profile also supports the use of UML <<access>> permissions that allow elements from the accessed packaged to be used within the accessing package, but require the use of qualified names such as Package1::X rather than X.

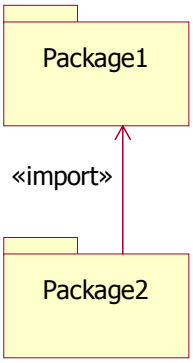


Figure 9: Notation for package and package dependency

Mapping to XSD/WSDL/BPEL

The package hierarchy of a model maps to an XML namespace hierarchy. The name of the model itself (as the top level package) also contributes to the namespace. The namespace prefix used in generated XML files is specified during model mapping. The namespace structure is also mirrored in the folder structure into which files are generated.

Packages containing elements that map to XSD/WSDL artifacts also map to files of the corresponding type. New files are not created for protocol packages, their elements are placed within the WSDL file corresponding to the parent package.

Each BPEL process maps to a BPEL definition in a BPEL file. Typically there is one BPEL process per package. Multiple BPEL processes can be placed in a single package, in which case multiple BPEL files are generated, but this is not recommended.

A package dependency maps to an XML namespace import.

Figure 10 shows the package structure of the purchase order example that was introduced in Section 0.

In this example the Purchase package leads to the generation of a Purchase.wsdl file. The PurchaseOrderProcess package contributes to the namespace of the PurchaseOrder.bpel file (generated due to the <<process>> class that it contains).

Note that the <<external>> stereotype indicates that no artifacts should be generated from the PurchaseDataTypes package.

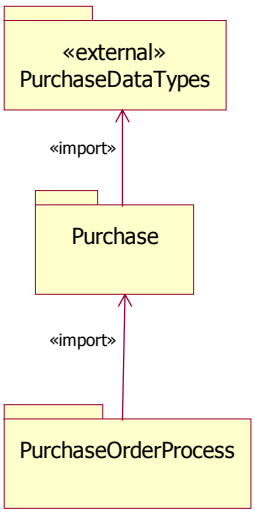


Figure 10: Package structure of the Purchase Order example

Figure 11 shows an extract from the WSDL file generated from the Purchase package. The target namespace prefix http://acme.org is specified during mapping. The name of the model (PurchaseOrder) and the name of the package (Purchase) provide the remainder of the namespace.

```
<definitions targetNamespace="http://acme.org/PurchaseOrder/Purchase"
      xmlns:PurchaseTypes="http://manufacturing.org/xsd/purchase"
      ...                                                          />

<import namespace="http://manufacturing.org/xsd/purchase"
        location="http://manufacturing.org/xsd/purchase.xsd"/>

<message name="POMessage">
   <part name="customerInfo" type="PurchaseTypes:CustomerInfo"/>
   <part name="purchaseOrder" type="PurchaseTypes:PurchaseOrder"/>
</message>
```

Figure 11: WSDL resulting from import of PurchaseTypes package into Purchase package

The PurchaseOrderProcess package contains a <<process>> class so a PurchaseOrderProcess.bpel file is generated. The imports of the PurchaseOrderProcess package are mapped to namespace imports in BPEL.

BPEL processes specify only the namespaces of their imports; the locations are provided at deployment time. BPWS4J requests this information through its administrative interface at deployment time[6].

```
<process name="PurchaseOrder"
   targetNamespace="http://acme.com/PurchaseOrder/PurchaseOrderProcess"
   xmlns="http://schemas.xmlsoap.org/ws/2002/07/business-process/"
```

[6] Platform-specific deployment artifacts could be generated during the mapping.

```
   xmlns:lns="http://acme.org/PurchaseOrder/Purchase">
```

Figure 12: BPEL namespace import

External packages

Concepts

A model containing one or more automated business processes can depend on elements that are defined outside of that model; such elements are referred to as external. For example, the purchase order example in section 0 uses a set of data types that are defined outside the purchase order model. Interfaces and protocols can also be defined externally and reused in the definition of a process.

External packages contain elements that are defined in another model or elements that are defined directly as platform-specific artifacts (such as Web services or BPEL documents). External packages are not mapped to platform-specific artifacts.

This profile does not currently provide a reverse mapping to enable existing artifacts to be automatically imported into a model for reuse. Elements that are reused can be modeled explicitly and placed in a package stereotyped with <<external>>. Figure 13 shows the use of the externally-defined PurchaseDataTypes package within the purchase order model.

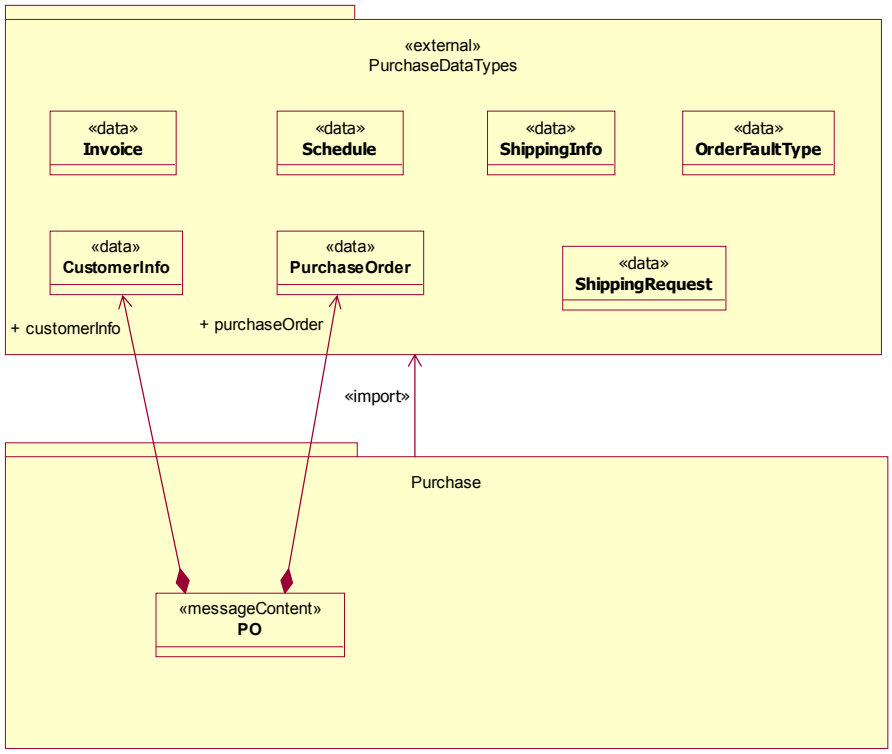


Figure 13: Use of external package.

Notation

The stereotype <<external>> is used to indicate that the elements in the PurchaseDataTypes package are defined outside of the current model and consequently should not be mapped to platform-specific artifacts.

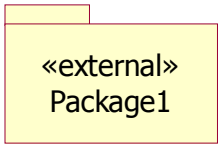


Figure 14: Notation for external packages

Mapping to XSD/WSDL/BPEL

External packages do not lead to the generation of artifacts. It is assumed that folders and files with names corresponding to those of the external packages will be accessible to the mapping tool. For example, the external artifacts are accessible within the classpath in a Java environment.

In Figure 15 the PurchaseDataTypes external package must have corresponding XSD artifacts in a PurchaseDataTypes.xsd file which is visible to the mapping tool.

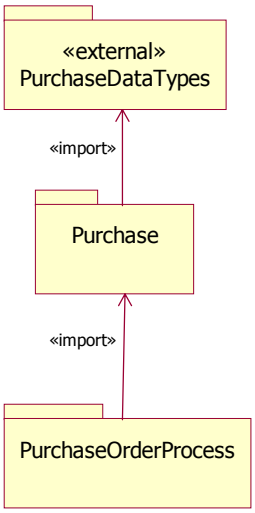


Figure 15: External package example.

Data types and interfaces

Concepts

The automated business process domain builds on the concepts and infrastructure provided by the Web services domain. A core concept for automated business processes is peer-to-peer communication between Web Services with both the process and the partners being described as Web Services. The Web Services concepts required for this profile are introduced in this document. It is anticipated that this profile will be aligned with a more general Web Services profile in the future.

A *port type* in Web services terminology defines a set of related operations. Port types are represented by interfaces. Operations take parameters that are typed by message types. A message type has a number of parts, each of which is of a specified *data type*. Data types

are represented by classes stereotyped as <<data>>; message types are represented by classes stereotyped as <<messageContent>>. This profile does not require message types to be introduced where they have only a single part; a data type can be used directly.

A data type should only be introduced where the grouping of attributes makes sense beyond its usage in a particular operation. Use messageContent where the grouping makes sense only in the context of an operation (or related set of operations).

Figure 16 shows a BuyerSeller package which contains two interfaces (Buyer and Seller) and three message types (BuyerInfo, Negotiation and SellerInfo). The optional dependency arrows indicate the data types that are used by each interface (these dependency arrows are not used by the mapping). In this example, message types are introduced to indicate that the groupings of attributes are used only for the purposes of packaging up operation inputs[7].

The attributes of a <<data>> class can have basic types or user-defined types. All of the attributes in the data types introduced in Figure 16 have basic types. The PO data type shown in Figure 16 has two parts which both have user-defined data types. Attributes of a <<data>> class are mapped to XSD attributes, containment relationships are mapped to XSD elements.

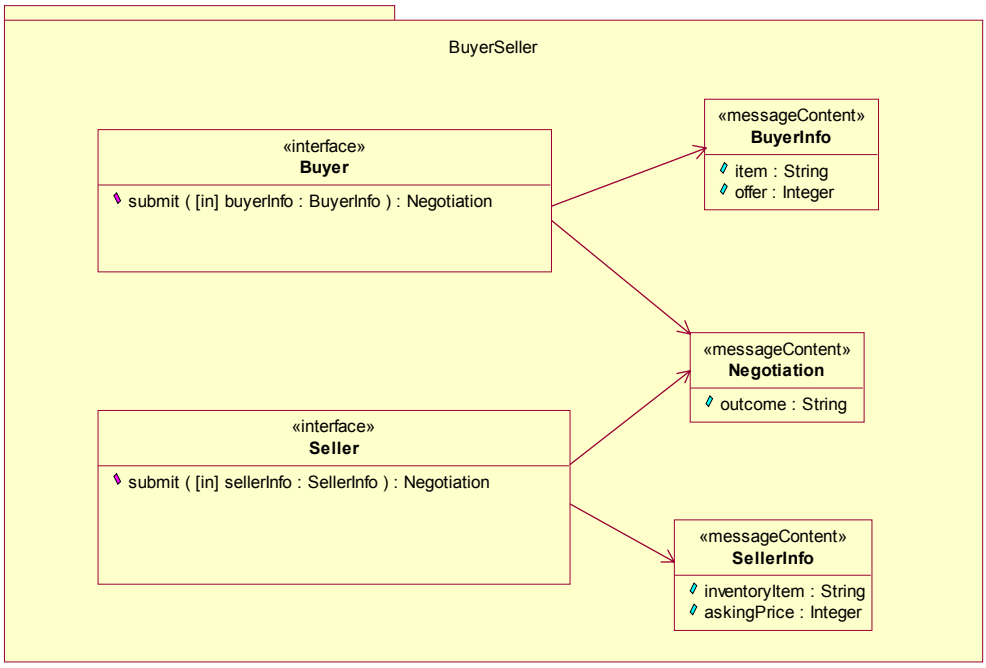


Figure 16: Interfaces and data types

[7] It is necessary to distinguish between the same grouping of parameter types occurring by accident and those groupings that represent uses of the same message type. It is important that this distinction can be made, because the mapping is to a messaging paradigm. A future version of this profile could enable users to specify operations with multiple parameters and to generate the corresponding message types.

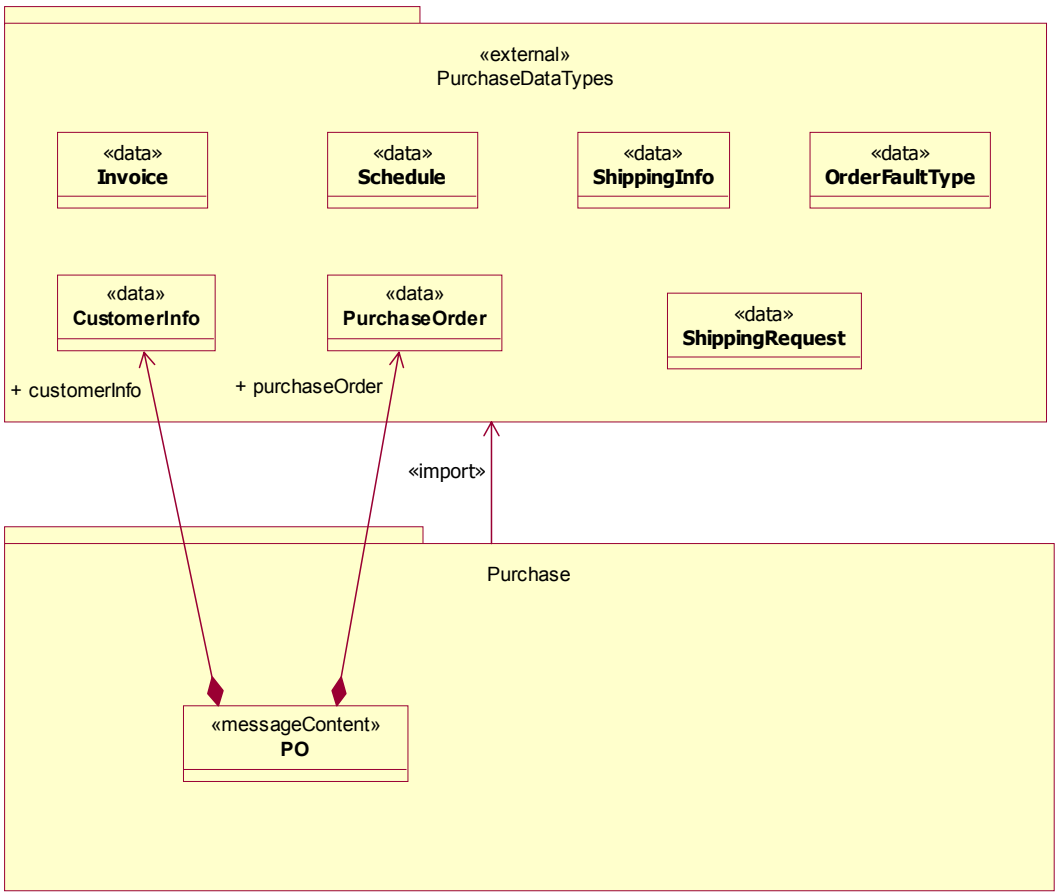


Figure 17: Parts with user defined types

Notation

Figure 18 shows the notation for interfaces, data types, and message types.

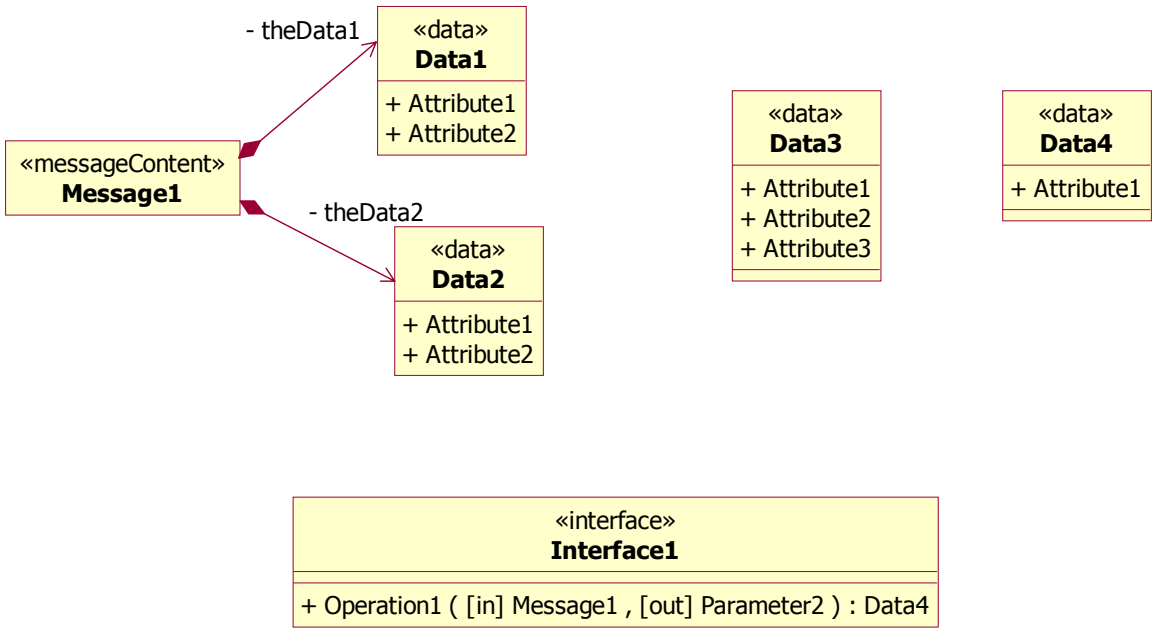


Figure 18: Notation for interfaces, data types, and message types

Inheritance between interfaces is not supported in this version of the automated business process profile[8]. Single inheritance between data types is supported via a mapping to XSD extensions.

Exceptions are represented by out parameters stereotyped as <<fault>>[9]. Note that the stereotype is not visible on the diagram in Rational XDE (see "Tool tips" for information about how to specify faults using Rational Rose). This enables you to use the Web services or BPEL approach to fault handling, in which faults are named – the name of the parameter provides the name of the fault.

Tool tips

RATIONAL XDE

An operation signature can be expanded on a class diagram, but the <<fault>> stereotype is not visible. It can be accessed through the properties view.

RATIONAL ROSE

[8] Note that inheritance is not supported in the current version of the WSDL standard, so support at the modeling level would need to be mapped to a single layer in a platform-specific model. This is not yet supported.

[9] When using the Java profile for Rational Rose or XDE, you can specify exception types that can be thrown from an operation. This profile is intended to be used with basic UML models that do not have this feature. Additionally, Java-style exceptions are specified by type only, but Web services and BPEL faults can be named.

Rational Rose does not support stereotyping of parameters with <<fault>>, and the Unisys XMI exporter does not distinguish between in and out parameters.

A naming convention is used to indicate parameters that are faults. If the name of a parameter, or the name of its type ends with 'Fault' then the parameter is treated as a fault.

For example, a parameter named processingFault with type Response is treated as a fault and so is a parameter named error with type LoanApprovalFault.

Mapping to WSDL and XSD

Each package containing interfaces is mapped to a separate WSDL document. Each interface within the package is mapped to a WSDL port type in the corresponding WSDL document.

Any class that has the stereotype <<data>> is mapped to an XSD type. Any <<messageContent>> class is mapped to a WSDL message type.

WSDL does not permit data types defined in XSD to be used directly as parameter types for operations. A message type containing multiple parts (each of which is of an XSD type) must be introduced.

This profile does not require that a new class is introduced to represent a message type containing a single part – the class representing the part type can be used directly. If a <<data>> class is used directly as the type of a process attribute, then a corresponding WSDL message type is introduced in the WSDL file for the package where the <<data>> class is used. (Repeated use of the same <<data>> class within a model does not lead to multiple message types.)

Additional classes can be introduced, if required; for example, to distinguish between different uses of the same type that should be mapped to different WSDL message types.

Figure 19 and Figure 20 show the definition of an interface and message type and the corresponding WSDL elements. Note that the [out] parameter represents a fault (the <<fault>> stereotype is not visible on the diagram).

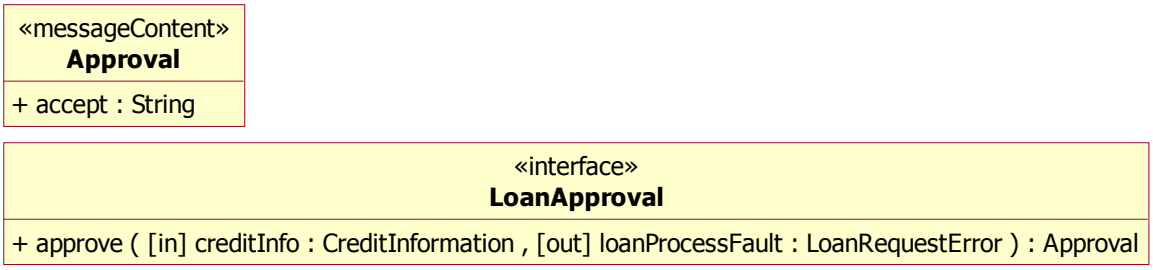


Figure 19: Interface and message type example

```
<wsdl:message name="Approval">
    <wsdl:part name="accept" type="xsd:string"/>
</wsdl:message>
<wsdl:portType name="LoanApproval">
    <wsdl:operation name="approve">
        <wsdl:input message="LoanDefinitions:CreditInformation"/>
        <wsdl:output message="tns:Approval"/>
```

```
            <wsdl:fault message="LoanDefinitions:LoanRequestError"
name="loanProcessFault"/>
        </wsdl:operation>
    </wsdl:portType>
```

Figure 20: WSDL port type and message type

Properties and correlations

Concepts

With loosely-coupled communicating business processes, explicit object references are not used to identify process instances. Instead, values contained within an incoming request are used to identify a process instance, or create a new instance if there is no existing instance matching those values. This is referred to as a *correlation*. A set of values used to identify an instance in a correlated exchange is referred to as a *correlation set*.

The messages within a correlated exchange must carry the values of the corresponding correlation set. However, the values can appear in different locations within the message. For example, instances of a Marketplace process providing a mediated negotiation can be identified by a correlation set containing a negotiation identifier. A message from a buyer can place this information in a negotiatedItem attribute, but the seller might place it in the 'item' attribute of a message. For this reason, correlation sets are defined in terms of properties. Properties provide an abstraction of a piece of information that is contained in many messages but may appear in a different element of each message. Properties can be used to correlate incoming messages with process instances.

A property has a name and a type. For each message containing the property, a property alias specifies how to locate the property in the message.

Any message that is sent or received as part of a correlated exchange must contain correlation information.

Figure 21 shows the definition of a negotiatedItem property of type String. Two property aliases are defined, which specify how to extract the negotiatedItem from a BuyerInfo and SellerInfo message. The computations performed by these aliases are very simple in this case: using XPath syntax, `/item` and `/inventoryItem` (the alias expressions are not visible on the diagram but are present in the definitions of the operations).

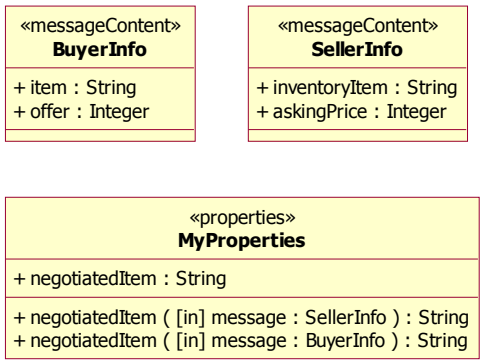


Figure 21: Negotiated item property definition.

A process can specify one or more correlation sets by which its instances can be identified. Figure 22 shows a correlation set containing one property (the negotiatedItem property introduced in Figure 21) that is used by the Marketplace process. The Negotiationclass stereotyped <<correlation>> represents the correlation set, and the attribute stereotyped <<correlation>> represents a usage of that correlation set by the process.

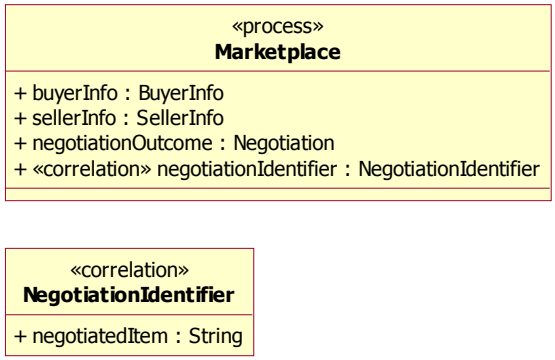


Figure 22: Correlation example

Figure 23 shows a receive activity within a correlated exchange that initializes the negotiatedItem correlation set.

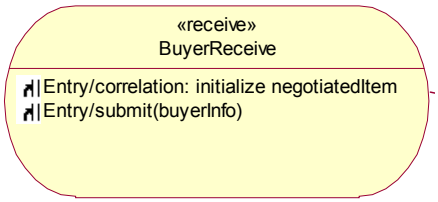


Figure 23: Correlation

Notation

Property definitions are introduced in a class stereotyped with <<properties>>. Each property is specified as an attribute with a name and type corresponding to that of the property. Property aliases are modeled as operations with the same name as the property and a return type corresponding to the type of the property. A property alias operation has a single input parameter of the type of message from which it extracts the property. An XPath expression is associated with the operation, specifying how the property is computed from the message. The exact way in which the XPath expression is associated with the operation depends on the tool being used (see "Tool tips").

The MyProperties class in Figure 24 is a <<properties>> class that introduces two properties: property1 and property2. Two aliases are defined for each property. For example, there is an alias for extracting property1 from messages of type MessageType1. The name of the class, MyProperties, is not used in the mapping, the class is simply used as a grouping mechanism for properties and property aliases.

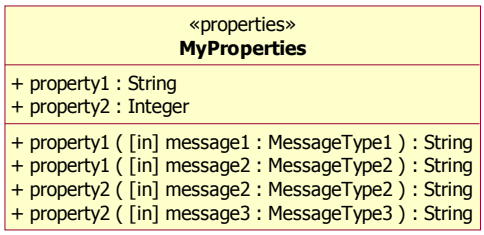


Figure 24: Notation for properties and property aliases

Property aliases can be defined in a package other than the one in which the property is defined. This is useful if a particular solution requires additional aliases for an existing property. In this case the aliases are placed in a subclass of the class that contains the property definitions.

A correlation set is defined by a class stereotyped by <<correlation>> containing attributes with names and types matching those of properties defined within its namespace. A process specifies that it uses a correlation set through an attribute with the type of the correlation set. The stereotype <<correlation>> can also be applied (redundantly) to the attribute to distinguish it from other attributes. Figure 25 shows the definition and use of a correlation set with two properties.

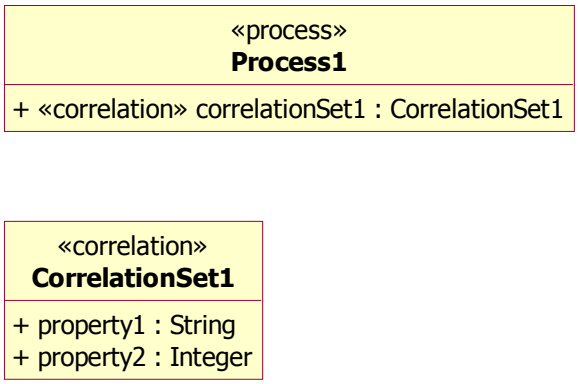


Figure 25: Notation for definition and use of a correlation set

In the simplest case, all messages coming into a process and going out of a process carry the same correlation set which is initialized at the start of the process. In this case you only need to associate a correlation set with the process as shown in Figure 25.

In other cases correlation sets need to be specified for individual invoke, receive and reply activities. This is achieved by placing a correlation expression in an entry action of a correlated activity as shown in Figure 26. This is in addition to the entry action containing the receive expression for the activity. Only interaction activities (receive, replay, and invoke) can contain correlation expressions.

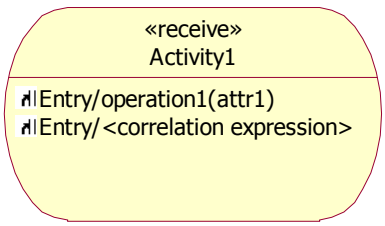


Figure 26: Notation for a correlated activities

Correlation expressions are prefixed by `correlation:`. A correlation expression indicates the correlation set, or sets that are used by the activity. Multiple correlation sets can be specified as a comma separated list.

Examples:

- correlation: invoiceID
- correlation: invoiceID, purchaseOrderNumber

If an activity is the first activity in the process to use a particular correlation set then it must specify that it can[10] initialize the correlation set. This is achieved by placing the keyword `initialize` before the correlation set.

Examples:

- correlation: initialize negotiationID
- correlation: invoiceID, initialize purchaseOrderNumber

In the case of invoke the request and returned response can carry separate correlation sets. A separate entry action is used for each direction. The keyword `out` is placed at the beginning of a correlation expression that refers to an outgoing request, and the keyword `in` is placed at the beginning of a correlation expression referring to an incoming response.

Examples:

- out correlation: purchaseOrderNumber
- in correlation: initialize invoiceID

Tool tips

RATIONAL XDE

In Rational XDE, *defaultExpression* for the return type of the operation is used to hold an XPath expression. That XPath expression describes how the property alias represented by the operation is computed.

RATIONAL ROSE

In Rational Rose, the *semantics* element of an operation used to hold an XPath expression. That XPath expression describes how the property alias represented by the operation is computed.

Mapping to WSDL or BPEL

[10] BPEL allows multiple concurrent activities to support initialization of a particular correlation set, the first one to occur initializes the correlation set and the other activities are then followers.

Each attribute within a <<properties>> class maps to a property definition within the WSDL file corresponding to the package where it is defined. BPEL properties are defined used WSDL extension elements and are therefore placed in a WSDL file rather than a BPEL file.

Each operation within a <<properties>> class maps to a property alias definition within the corresponding WSDL file.

The <<properties>> class in Figure 21 maps to the BPEL (WSDL extensions) shown in Figure 27. Note that the queries are not visible on the diagram, but do appear in the model.

```
<bpws:property name="negotiatedItem" type="xsd:string"/>

<bpws:propertyAlias propertyName="tns:negotiatedItem"
                    messageType="tns:SellerInfo"
                    part="inventoryItem"
                    query="/inventoryItem"/>

<bpws:propertyAlias propertyName="tns:negotiatedItem"
                    messageType="tns:BuyerInfo"
                    part="item"
                    query="/item"/>
```

Figure 27: BPEL property and property aliases

Each <<correlation>> attribute on a process maps to the definition of a correlation set in the BPEL file for that process, the details of the correlation set come from the <<correlation>> class that types the attribute.

The correlation set defined in Figure 22 maps to the BPEL in Figure 28.

```
<correlationSets>
  <correlationSet name="negotiationIdentifier"
                  properties="tns:negotiatedItem"/>
</correlationSets>
```

Figure 28: BPEL for a correlation set

Correlation expressions on activities map to correlation expressions in the corresponding activities.

The correlated activity shown in Figure 23 maps to the BPEL in Figure 29.

```
    <receive partner="buyer" portType="tns:buyerPT"
             operation="submit" variable="buyerInfo"
             createInstance="yes" name="BuyerReceive">
      <correlations>
        <correlation set="negotiationIdentifier" initiation="yes"/>
      </correlations>
    </receive>
```

Figure 29: BPEL correlated activity

Protocols

Concepts

The relationship between a business process and partner can need messages to flow in either or both directions.

A protocol links together two complementary roles, such as Bidder and AuctionHouse and specifies the interfaces that are provided or needed by each role.

A process has a number of ports, with each port corresponding to a role in a protocol. The role determines the interfaces that are required and provided through that port.

Figure 30 show four protocols. Purchase and Scheduling are one-way protocols. A process playing one of the roles in these protocols either sends messages to a partner playing the complementary role, *or* receives messages from the partner. Invoice and Shipping are two-way protocols in which both parties both send and receive messages.

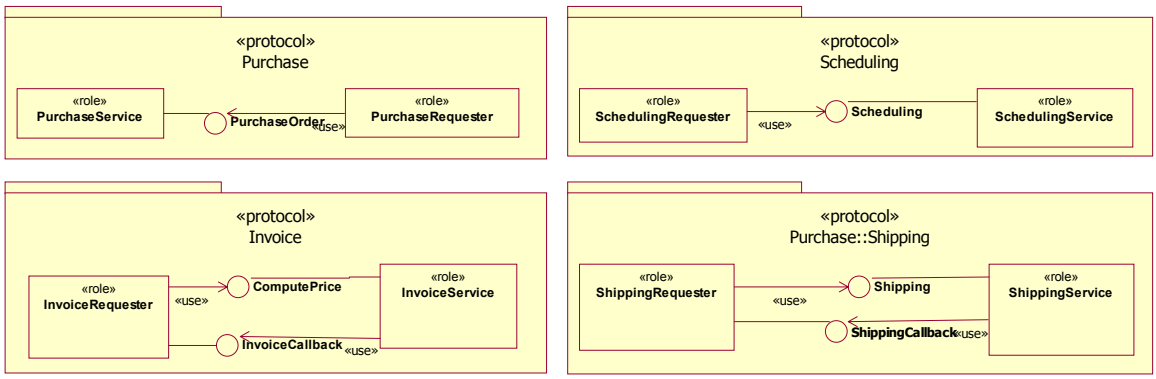


Figure 30: Protocols and Roles

In addition to showing the structuring of the roles, it is also valuable to show the valid sequences of message exchanges between roles. This can be shown using one or more interaction (sequence) diagrams (collaboration diagrams can also be used). The profile currently does not require such sequence diagrams, but they can provide valuable additional documentation for a human user of the profile, and could be used for validation purposes[11].

Figure 31 shows an interaction diagram for the Invoice protocol[12]. In this case there is only one valid message exchange sequence so a single interaction diagram is sufficient to capture all valid sequences.

The lifelines represent instances of the roles defined for the protocol.

---

[11] An interaction diagram could be used to generate abstract processes for both roles. An executable process that is playing one of the roles would then need to conform to the corresponding abstract process. An interaction diagram can also provide a starting point for a BPEL process that would need to have activities corresponding to the valid message sends and receives for its role.

[12] The diagram was drawn using Rational XDE, which uses the half arrowhead for asynchronous invocation. In Rational Rose, you can also use the full open arrowhead, which also means asynchronous.

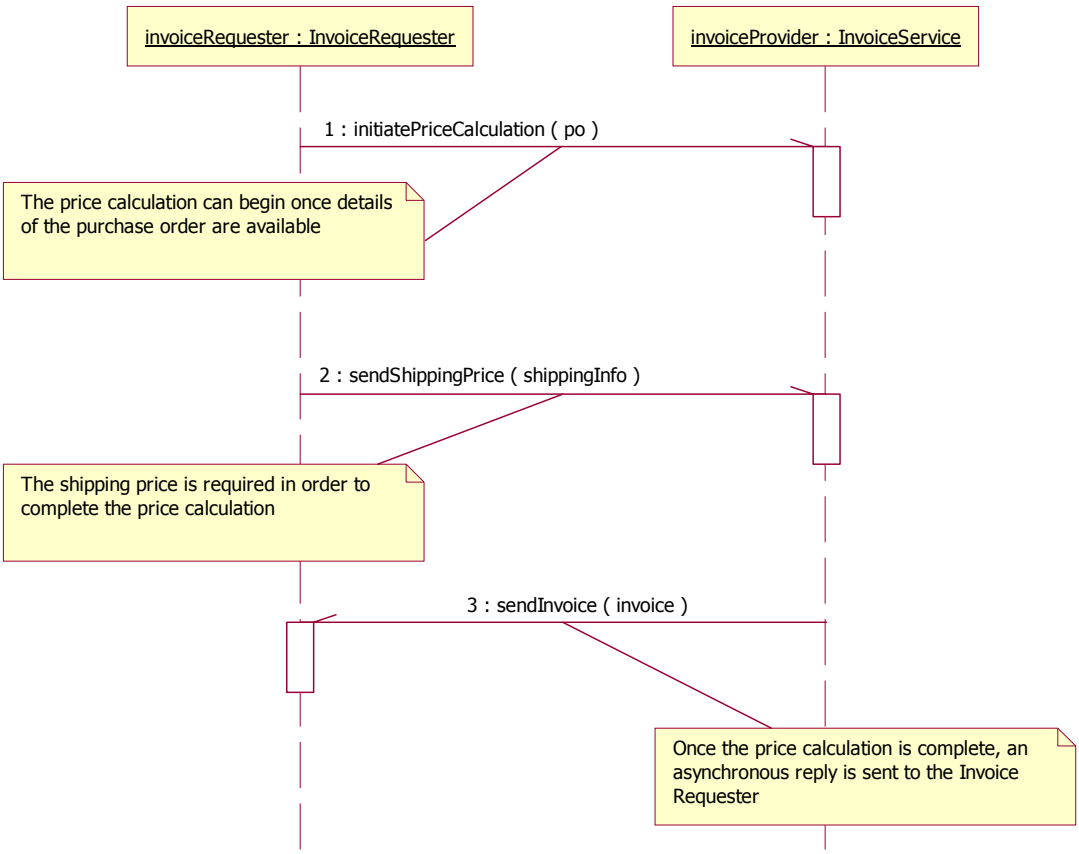


Figure 31: Interaction diagram for the Invoice protocol

Notation

A protocol is modeled as a package stereotyped as <<protocol>>. A protocol contains a pair of classes stereotyped as <<role>> as shown in Figure 32. The name of the package is the name of the protocol.

Each role has a "realizes" dependency to the interfaces it provides and a one-way dependency to the interfacess it requires.

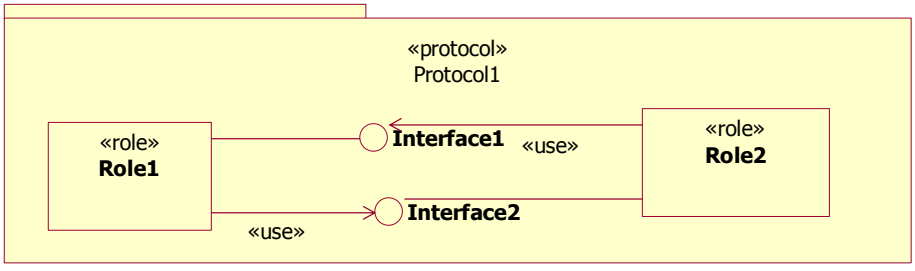


Figure 32: Protocol notation

Figure 32 shows a protocol named Protocol1. Role1 provides Interface1 and requires Interface2, and Role2 provides Interface2 and requires Interface1.

In one-directional protocols only one role provides an interface, which the other role requires.

A role can provide multiple interfaces in which case the corresponding role requires all of the interfaces. In the current version of the profile it is not possible to distinguish between operations with the same signature that occur in different interfaces provided by the same role.

Tool tips

RATIONAL XDE

Role classes do not introduce any attributes or operations so they can be shown without additional compartments. Select a role class in a diagram, then click **Format-> Compartments-> None** from the main menu.

To introduce an optional interaction diagram, navigate to the appropriate protocol package, then chose **Add Diagram-> Sequence: Instance**. To introduce lifelines that represent instances of roles, drag the role classes on to the diagram.

To adjust the length of lifelines, change the Height value in the properties menu for that lifeline.

Style guidelines

Class diagrams that show protocols can be displayed inside the box representing the package that corresponds to the protocol (if the tool being used supports this). When defining a group of related protocols, it can be convenient to place them in a common diagram, which should be placed in the package that contains the related protocol packages.

Mapping to BPEL

Each protocol package in the UML profile maps to a BPEL service link type. The roles in the protocol become roles in the service link type with interfaces provided, becoming port types within the roles.

Protocols with a one-way dependency are mapped to service link types with only one role.

The protocol shown in Figure 33 maps to the BPEL extract (actually a WSDL extension, because service link types are placed in a WSDL document) shown in Figure 34.

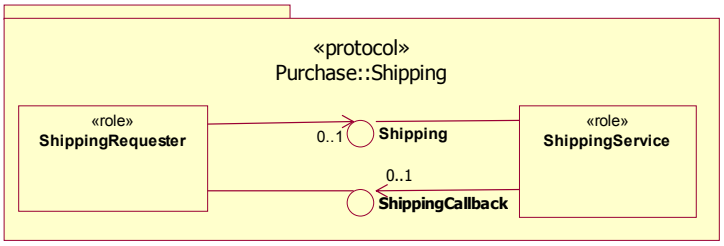


Figure 33: Shipping protocol

```
<slnk:serviceLinkType name="Shipping">
   <slnk:role name="ShippingService">
```

```
        <slnk:portType name="Purchase:Shipping"/>
    </slnk:role>
    <slnk:role name="ShippingRequester">
        <portType name="Purchase:ShippingCallback"/>
    </slnk:role>
</slnk:serviceLinkType>
```

Figure 34: BPEL (actually WSDL extension) for shipping service link type

Interaction and collaborations diagrams do not contribute to the mapping.

Process, state, and ports

Concepts

An automated business process describes the execution of a multi-party business-relevant task such as an auction or a purchase order. The process is stateful and interacts with a number of partners through defined protocols.

A process can be involved in many protocols with different partners, and it can be involved in the same protocol with multiple partners. When modeling an individual process the connections to partners are not modeled, rather the connections are made indirectly through ports. Each port is an interaction point of the process and specifies a role in a protocol that is supported by the process. The role defines the set of interfaces that are provided or required through that port.

UML 2 [] provides direct support for a port-connector model and corresponding notation as shown in Figure 35.

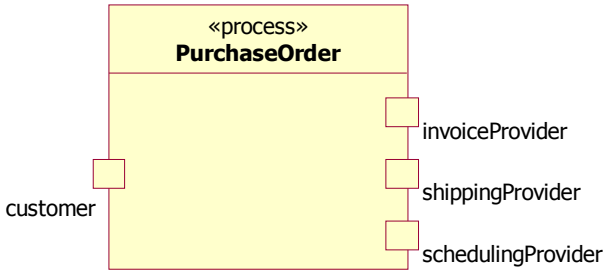


Figure 35: UML 2 notation for ports

The UML 2 notation is not available to this UML 1.x profile so an alternative is provided. Figure 36 shows a PurchaseOrder process with four ports. The process has state, recorded using five attributes.

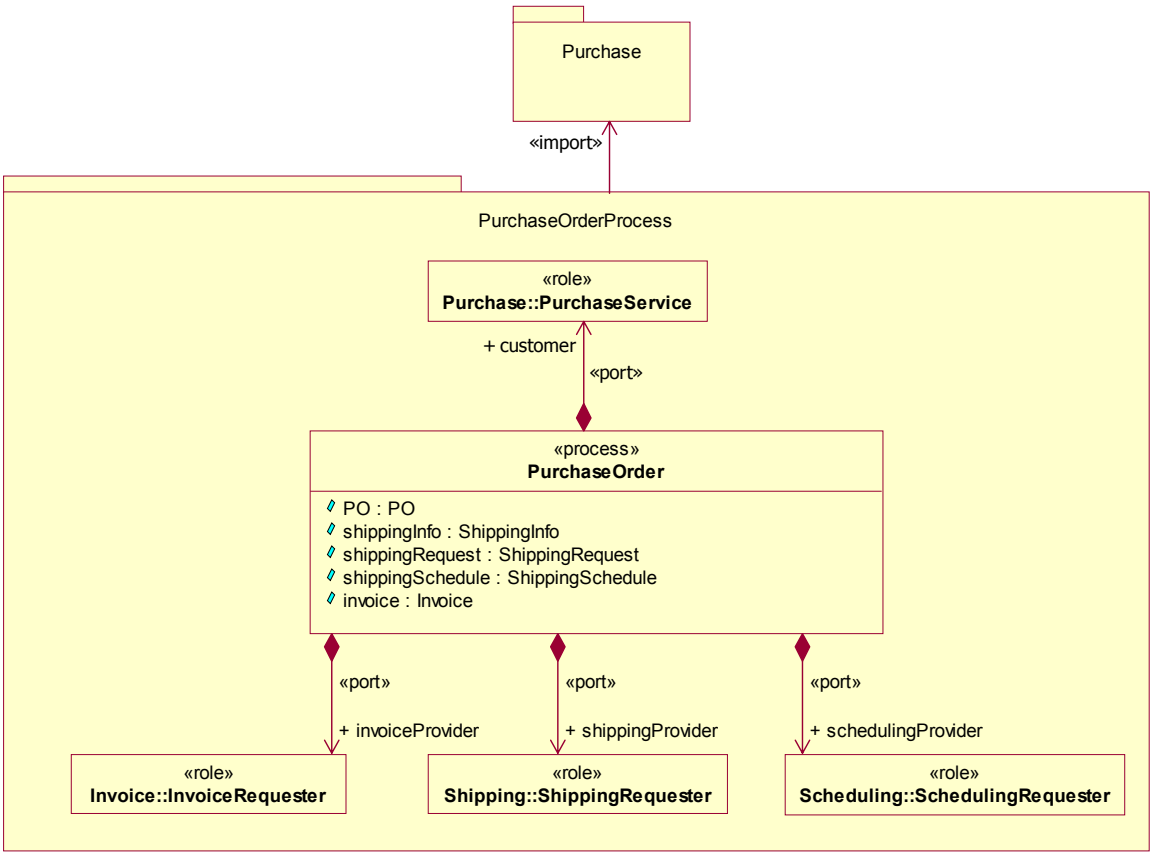


Figure 36: Ports of the PurchaseOrder process

Like in BPEL, a port can be connected to zero or one partners at any time. Support for broadcast ports is not supported in the current version of this profile.[13]

Notation

A process is modeled as a class stereotyped as <<process>>. The attributes of the class correspond to the state of the process. Each port is represented by an association, stereotyped <<port>>, to the role offered by that port.

Figure 37 shows a process named Process1 which maintains its state in attribute1 and attribute2. Process1 has two ports, port1 and port2, indicating that the process participates in ProtocolX (in Role1) and ProtocolY (in Role2).

The package name is the name of the protocol (for example, ProtocolX::Role1). If the protocol package is directly imported by the process package, you can omit the package name in the name of roles played by the process.

By convention, ports have public visibility, but this is not required for the mapping.

[13] Broadcast ports could be modeled in UML; for example, by using the multiplicity of the port association. This aspect of the profile would need to be mapped to another middleware component since BPEL does not support the notation of broadcast.

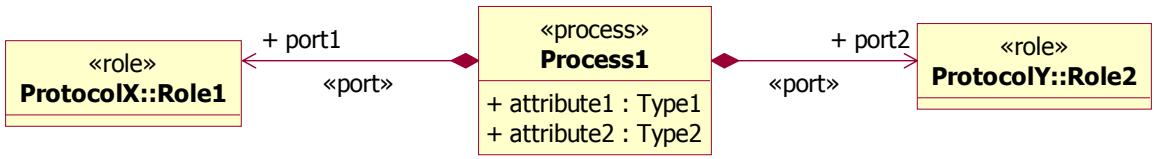


Figure 37: Notation for process with state and ports

Presentation options

When showing ports, the roles can be expanded to show the interfaces that are required and provided through each port. These are defined as part of the corresponding protocols, but it can be convenient to view all relationships on a single diagram. Figure 38 shows four ports with expanded roles. The customer port provides the PurchaseOrder interface and the schedulingProvider port requires the Scheduling interface. Both the invoiceProvider and shippingProvider ports both provide and require an interface.

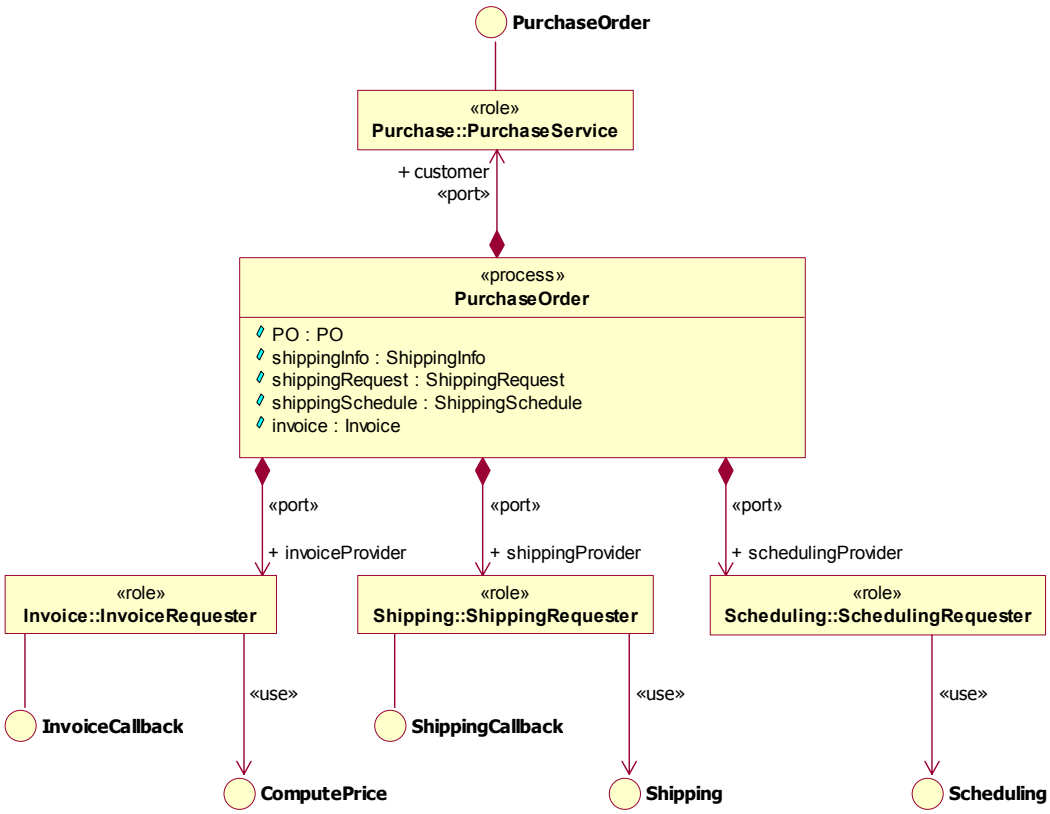


Figure 38: Provided and required interfaces shown on a process diagram

The examples above follow the convention of using attributes to represent variables and associations to represent ports. In fact, attributes or associations can be used in either case. An attribute representing a port must be stereotyped as <<port>>, and its type must be a class stereotyped as <<role>>. Note that when using attributes to represent ports in it not possible to show associations to provided and required interfaces as shown in Figure 38.

Tool tips

RATIONAL XDE

To add required and provided interfaces to a diagram, when using a role that has been defined as part of a protocol, right-click on a <<role>> class in a class diagram then click **Add Related Shapes**.

Mapping to BPEL

A class stereotyped as <<process>> when using this UML profile will be mapped to a BPEL process definition with the following attributes:

attribute	value	notes
abstractProcess	no	This profile supports the modeling of executable processes only.
variableAccessSerializable	no	This concept is not yet supported by this profile.
enableInstanceCompensation	no	Instance compensation after completion of the process requires platform specific support which is not yet available.
SuppressJoinFailure	yes	Join failures occur when the status of all incoming links to an activity is negative (using the default join condition). This is not considered to be an error condition in models developed using this profile.

The target namespace prefix of the process is provided during mapping. The remainder of the namespace is taken from the name of the model, the package structure and the name of the process class.

The attributes of the process map to BPEL variables and the ports (associations to roles) map to BPEL partner declarations. In the UML profile there is no need to specify the role played by the partner since this is always the complementary role that is defined in the same protocol as the role played by the process. The BPEL partnerRole comes from the name of the complementary role in the protocol.

Figure 39 and Figure 40 show a LoanApproval process with its corresponding BPEL.

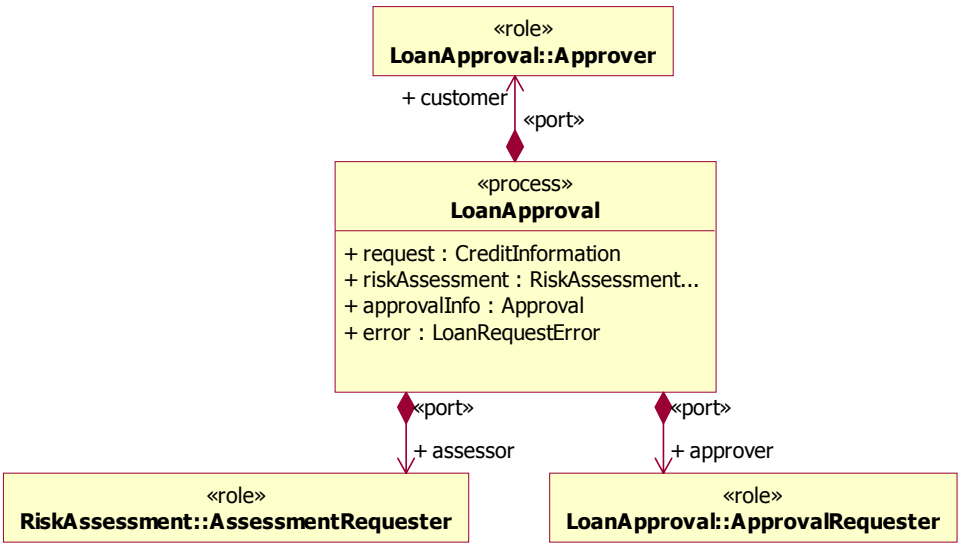


Figure 39: LoanApproval process

```
<process
    abstractProcess="no"
    variableAccessSerializable="no"
    enableInstanceCompensation="no"
    name="LoanApproval"
    suppressJoinFailure="yes"
    targetNamespace="http://www.bpel-examples.ibm.com/
LoanApproval/LoanApprovalProcess/LoanApproval.bpel">
    <partners>
        <partner myRole="Approver" name="customer"
                 serviceLinkType="LoanApprovalProtocols:LoanApproval"/>
        <partner name="approver" partnerRole="Approver"
                 serviceLinkType="LoanApprovalProtocols:LoanApproval"/>
        <partner name="assessor" partnerRole="Assessor"
                 serviceLinkType="LoanApprovalProtocols:RiskAssessment"/>
    </partners>
    <variables>
        <variable messageType="LoanDefinitions:CreditInformation"
                   name="request"/>
        <variable messageType="LoanAssessor:RiskAssessmentDetails"
                   name="riskAssessment"/>
        <variable messageType="LoanApprover:Approval"
                   name="approvalInfo"/>
        <variable messageType="LoanDefinitions:LoanRequestError"
                   name="error"/>
    </variables>
```

```
  …
</process>
```

Figure 40: Process Definition in BPEL

Each <<data>> class is mapped to an XSD type. Any <<data>> class that is used as the type of an attribute in a process is additionally mapped to a WSDL message type. The WSDL message type is placed in the same WSDL file as the corresponding <<data>> class. If the package is <<external>> then the WSDL message type is placed in a WSDL file containing WSDL elements scoped to the model (the namespace and location of which will be configurable during mapping).

Note that <<messageContent>> classes used as the type of an attribute have WSDL messages generated in the WSDL file corresponding to the package in which the <<messageContent>> class *is defined*. The corresponding message types are generated regardless of whether the types are actually used to type process attributes.

The RiskAssessmentDetails class used as the type of the riskAssessment variable is a <<data>> class. The WSDL in Figure 41 is generated into the WSDL file for the LoanAssessment package where RiskAssessmentDetails is defined.

```
    <wsdl:message name="RiskAssessmentDetails">
        <wsdl:part name="risk" type="xsd:string"/>
    </wsdl:message>
```

Figure 41: WSDL Message example

Data handling

Concepts

Within a business process, you need to be able to manipulate and update the state of the process. This is achieved through having an expression language.

Expressions are used in the following places within this profile:

- Boolean expressions are used in transition guards
- Time expressions are used as triggers in onAlarm events and in wait activities
- General expressions are used in assignment activities

Both UML and BPEL permit any scripting language to be used to write such expressions. BPEL uses XPATH as the default scripting language and provides conventions for accessing variable data and BPEL-provided utility functions from XPATH expressions. This version of the profile supports XPATH as the default language for expressions and provides conventions for accessing attributes. This provides a straightforward mapping to BPEL.

In future, other expression languages can be supported. In particular, if a standard surface language for the UML activity and action semantics is agreed upon then we would expect to support that.

Notation and mapping to XPATH or BPEL

ACCESSING VARIABLES AND PROPERTIES

Within expressions you need to be able to refer to elements of the state of a process. You refer to state elements through XPATH expressions rooted to the process.

Each expression begins with the name of an attribute of the process. Navigation to a part of the message, or a property of the message is achieved by appending a forward slash followed by the part or property name. Further navigation into the internal structure of a part can be specified by appending additional XPATH elements to the expression.

For example, the expression

```
shippingRequest/customerInfo
```

in a purchase order process, refers to the customerInfo attribute of the shippingRequest attribute of the purchase order process.

MAPPING TO XPATH/BPEL

The BPEL specification defines the following functions for use within XPATH expressions:

```
bpws:getVariableData ('variableName', 'partName', 'locationPath'?)
```

This function extracts values from variables. The first argument names the source variable for the data, the second names the part to select from that variable, and the third optional argument, when present, provides an absolute location path (with '/' meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part. The return value of this function is a node set containing the single node representing either an entire part (if the third argument is absent) or the result of the selection based on the locationPath. If the given locationPath selects a node set of a size other than one, then the standard fault `bpws:selectionFailure` MUST be thrown by a compliant implementation.

```
bpws:getVariableProperty ('variableName', 'propertyName')
```

This function extracts global property values from variables. The first argument names the source variable for the data and the second is the qualified name (QName) of the global property to select from that variable … If the given property does not appear in any of the parts of the variable's message type, then the standard fault `bpws:selectionFailure` MUST be thrown by a compliant implementation. The return value of this function is a node set containing the single node representing the property. If the given property definition selects a node set of a size other than one, then the standard fault `bpws:selectionFailure` MUST be thrown by a compliant implementation.

Data access expressions are mapped to uses of these functions. The first element of a data access expression is always an attribute, which maps to the first argument in either a getVariableData or a getVariableProperty call.

The second element in a data access expression is used to determine whether the expression is to be mapped to a use of getVariableData or getVariableProperty. This depends on whether the element corresponds to a part or a property of the attribute.

If the second element is a part then there can be more elements that map to a location path. (The location path must be an absolute path from the part, so it also includes the part name as the first element.)

If a message has a part and a property alias for a property with the same name as that part, then the property alias must refer to that part. Such as expression is mapped to a getVariableProperty call unless, it has elements following the second element which map to a location path, in which case a getVariableData call be be generated.

Examples:

request/amount where: request is an attribute of type Request which has an attribute named amount	bpws:getVariableData('request','amount')
stockResult/level where: stockResult is an attribute of type StockResult for which a property alias is defined for the level property	bpws:getVariableProperty(stockResult,level)
customer/address/name/last where: customer is an attribute and the remainder of the expression navigates its internal structure The expression extracts the value of the 'last' attribute.	bpws:getVariableData('customer', 'address', '/address/name/last')

GENERAL EXPRESSIONS

As in the BPEL specification: "These are expressions that conform to the XPath 1.0 Expr production where the evaluation results in any XPath value type (string, number, or Boolean).

Expressions with operators are restricted as follows:

- All numeric values including arbitrary constants are permitted with the equality or relational operators (<, <=, =, !=, >=, >).
- Values of integral (short, int, long, unsignedShort, and so on) type including constants are permitted in numeric expressions, provided that only integer arithmetic is performed. In practice, this means that division is disallowed. It is difficult to enforce this restriction in XPath 1.0 because XPath 1.0 lacks integral support for types. The restriction should be taken as a statement of intent that will

be enforced in the future when expression languages with more refined type systems become available.

- Only equality operators (=, !=) are permitted when used with values of string type including constants." []

MAPPING TO XPATH OR BPEL

General expressions are mapped to XPATH expressions, with the contained data access expressions being mapped as described above.

Examples:

responses/expected – responses/received	bpws:getVariableData('responses', 'expected') – bpws:getVariableData('responses', 'received')

BOOLEAN EXPRESSIONS

As in the BPEL specification: "These are expressions that conform to the XPath 1.0 Expr production where the evaluation results in Boolean values." []

Examples:

```
buyer/offerPrice >= seller/reservePrice
```

MAPPING TO BPEL

Boolean expressions are mapped as for general expressions.

Examples:

request/amount >= 10000	bpws:getVariableData('request', 'amount') >= 10000
riskAssessment/risk = 'low'	bpws:getVariableData('riskAssessment', 'risk') = 'low'

TIME EXPRESSIONS

DEADLINE-VALUED EXPRESSIONS

As in the BPEL specification: "These are expressions that conform to the XPath 1.0 Expr production where the evaluation results in values that are of the XML Schema types *dateTime* or *date*." []

DURATION-VALUED EXPRESSIONS

As in the BPEL specification: "These are expressions that conform to the XPath 1.0 Expr production where the evaluation results in values that are of the XML Schema type *duration*." []

MAPPING TO BPEL

Time expressions are mapped directly to time expressions in BPEL. For an example, see Section 0 where the mapping of wait activities to BPEL is covered.

ASSIGNMENT

Assignment statements are represented by using the := operation between general expressions. This indicates that the value of the element on the right-hand side of the expression is copied into the element specified on the left-hand side.

For example, the assignment statement:

```
shippingRequest/customerInfo := PO/customerInfo
```

copies the value of the customerInfo attribute of the PO attribute into the customerInfo attribute of the shippingRequest attribute.

MAPPING TO BPEL

An assign activity maps to a BPEL assign activity with each entry action mapping to a copy element. The right-hand side of each assign statement provides the details of a from element and the left-hand side provides then details of the to element.

Figure 42 and Figure 43 show an <<assign>> activity and its corresponding BPEL.

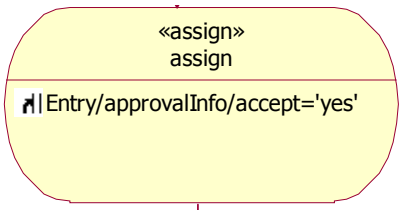


Figure 42: Example assign activity

```
<assign name="assign">
    <copy>
        <from expression="'yes'"/>
        <to variable="approvalInfo" part="accept"/>
    </copy>
</assign>
```

Figure 43: BPEL assign activity

Basic activities

Invoking operations through ports

CONCEPTS

A process often invokes operations in its partners as part of its execution. This can be done synchronously or asynchrously:

- Synchronously. A fault can be thrown and a return value can be provided
- Asynchronously. The activity completes as soon as the request has been issued and there is no corresponding return or fault.

An asynchronous reply is modeled as a further one-way invocation from receiver to caller.

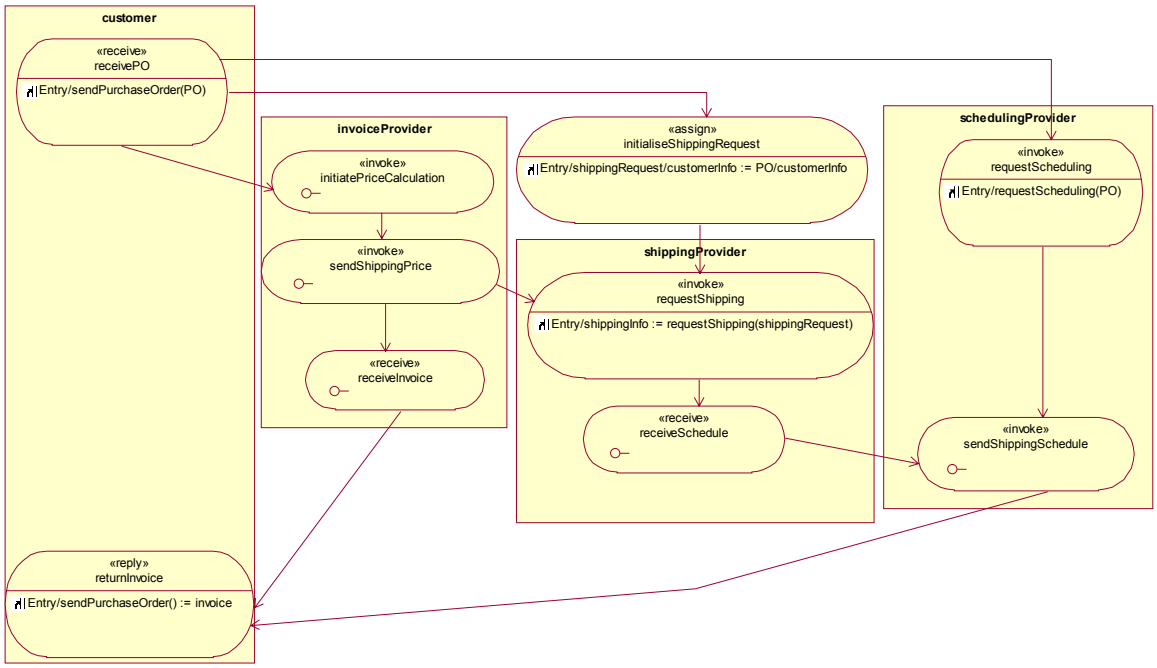


Figure 44 Illustration of invoke and receive/reply activities

Figure 44 (which is a repeat of Figure 7) contains a number of invoke activities. A synchronous invocation is exemplified by the requestShipping activity which invokes the requestShipping operation through the shippingProvider port. An asynchronous invocation is exemplified by the requestScheduling activity, which invokes the requestProductionScheduling operation through the schedulingProvider port.

NOTATION

An invocation of an operation on a partner is represented as an activity with stereotype <<invoke>> with an entry action indicating the operation to be invoked and the attribute containing the input message. For two-way messages, assignment notation is used to indicate the attribute that is updated with the reply message.

Communication with partners is through ports. So the partner on which the operation is invoked is specified by placing the activity in the partition corresponding to the port through which communication with the partner takes place. Note that, as in BPEL, a port can be connected to only one partner so an invoke activity causes an operation to be invoked on one partner.

Figure 45 shows two invoke activities. Activity1 is a one-way activity (it does not specify an attribute for a result) which invokes operation1 through port1 with attr1 as the input message. Activity1 completes as soon as the operation invocation request has been sent and does not wait for a reply. Activity2 is a two-way invoke which invokes operation2

through port2 with attr2 as the input message and waits for a synchronous reply which it places in attr3.

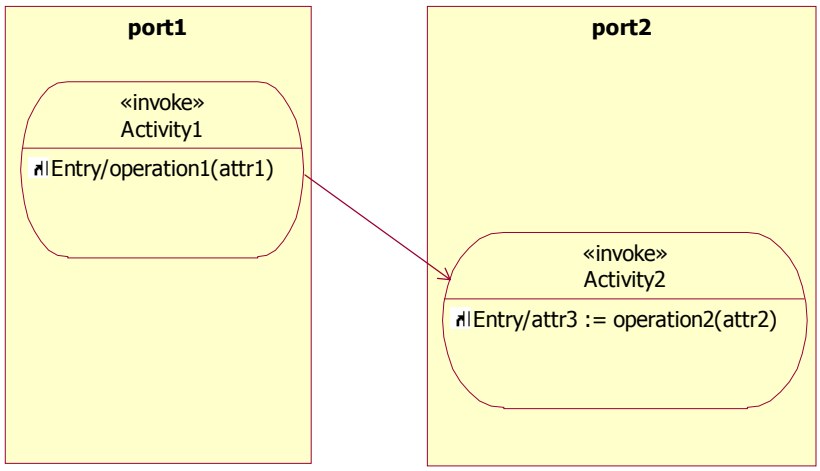


Figure 45: Invoke notation

When using a hierarchical structure to denote control flow, it is not possible in Rational XDE or Rational Rose to place nested activities in partitions. In this case, use the alternate notation for partitions that is provided in UML 2: the name of the partition is placed in round brackets above, or in front of, the activity name as shown in Figure 46.

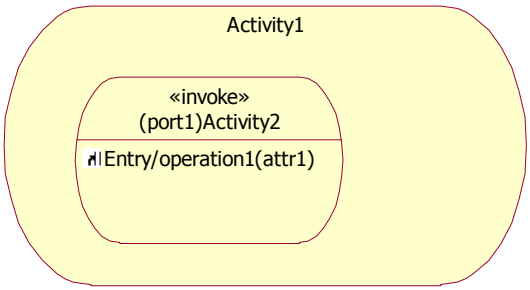


Figure 46: Alternate notation for nested interaction activities

MAPPING TO BPEL

Invoke activities are mapped to BPEL invoke activities. The name of the UML activity is used for the name of the BPEL activity. The port name becomes the name of the partner. The operation name and optional input and output variables come from the invoke expression in the entry action of the activity. The port type corresponds to the name of the required interface on the port that contains the operation.

Figure 47 shows the BPEL generated from the invokeAssessor and invokeApprover activities in Figure 48. Because the BPEL activities are contained within a flow, the activities also contain information on the links in which they are the source or target.

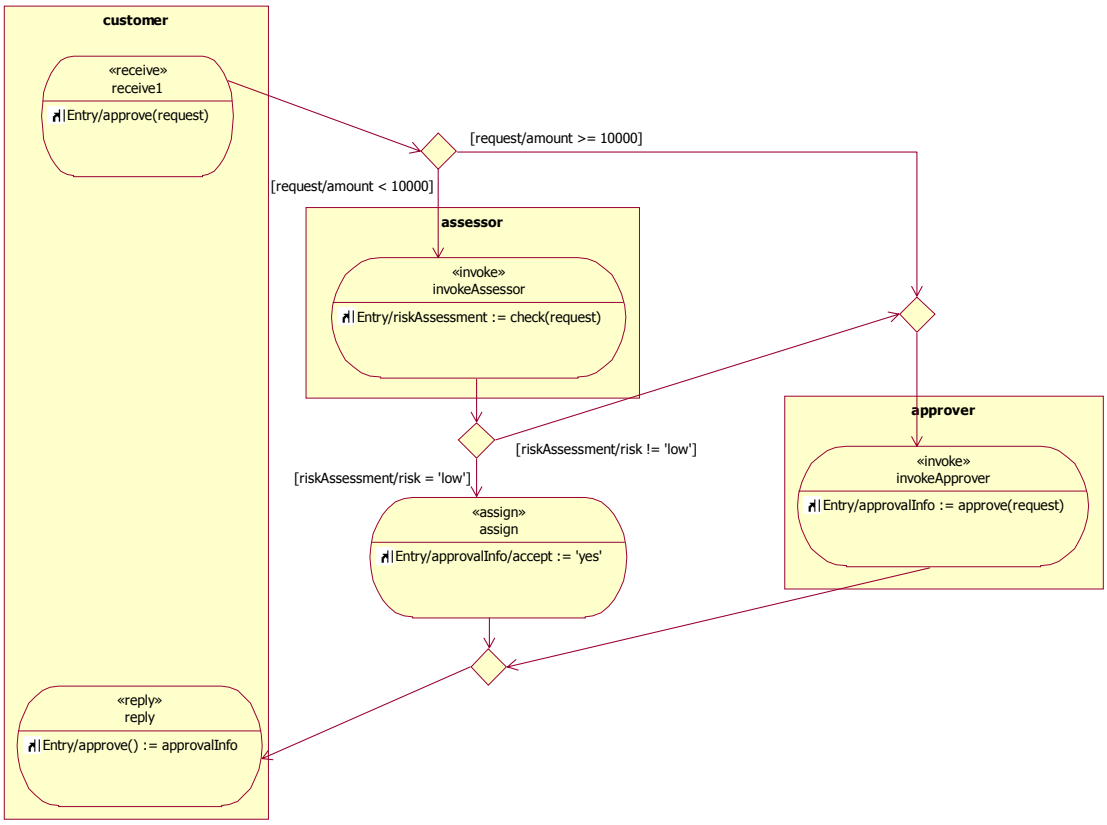


Figure 47: Loan Approval example with invoke activities

```
         <invoke inputVariable="request" name="invokeAssessor"
            operation="check" outputVariable="riskAssessment"
            partner="assessor" portType="LoanAssessor:RiskAssessment"
suppressJoinFailure="no">
            <target linkName="receive1_to_invokeAssessor"/>
            <source linkName="invokeAssessor_to_invokeApprover"
transitionCondition="bpws:getVariableData('riskAssessment','risk') != 'low'"/>
            <source linkName="invokeAssessor_to_assign"
transitionCondition="bpws:getVariableData('riskAssessment','risk') = 'low'"/>
        </invoke>

        …
        <invoke inputVariable="request" name="invokeApprover"
            operation="approve" outputVariable="approvalInfo"
            partner="approver" portType="LoanApprover:LoanApproval"
suppressJoinFailure="no">
            <target linkName="receive1_to_invokeApprover"/>
            <target linkName="invokeAssessor_to_invokeApprover"/>
            <source linkName="invokeApprover_to_reply"/>
```

```
        </invoke>
```

Figure 48: BPEL invoke activities

Providing operations through ports

CONCEPTS

A process provides operations to its partners through *receive activities*. Synchronous replies are modeled using corresponding *reply activities* and the caller is assumed to block. Asynchronous requests do not have corresponding reply activities.

Examples of receive and reply activities are provided by Figure 44 on page 77. An asynchronous request is exemplified by receiveInvoice from InvoiceProvider, and a synchronous request/reply is illustrated by receivePO and returnInvoice from or to Customer.

NOTATION

Receive activities have the stereotype <<receive>> and an entry action, which indicates the operation call that is expected and the attribute into which the input message will be placed. The partner that the operation call is expected from is specified by placing the activity in the partition that corresponds to the port through which the partner communicates.

Figure 49 shows a receive/reply pair of activities (Activity1 and Activity4) modeling the receipt of a synchronous request and the corresponding reply, and the receipt of a one-way operation call (Activity3) which has no corresponding reply.

When it receives control, Activity1 waits for the receipt of an invocation of operation1 from port1. When Activity1 receives that invocation, the input message is placed in attr1 and Activity1 completes.

When control reaches Activity4, attr2 is returned as the result of operation1. As in BPEL, there can be only one request outstanding for a particular partner and operation.

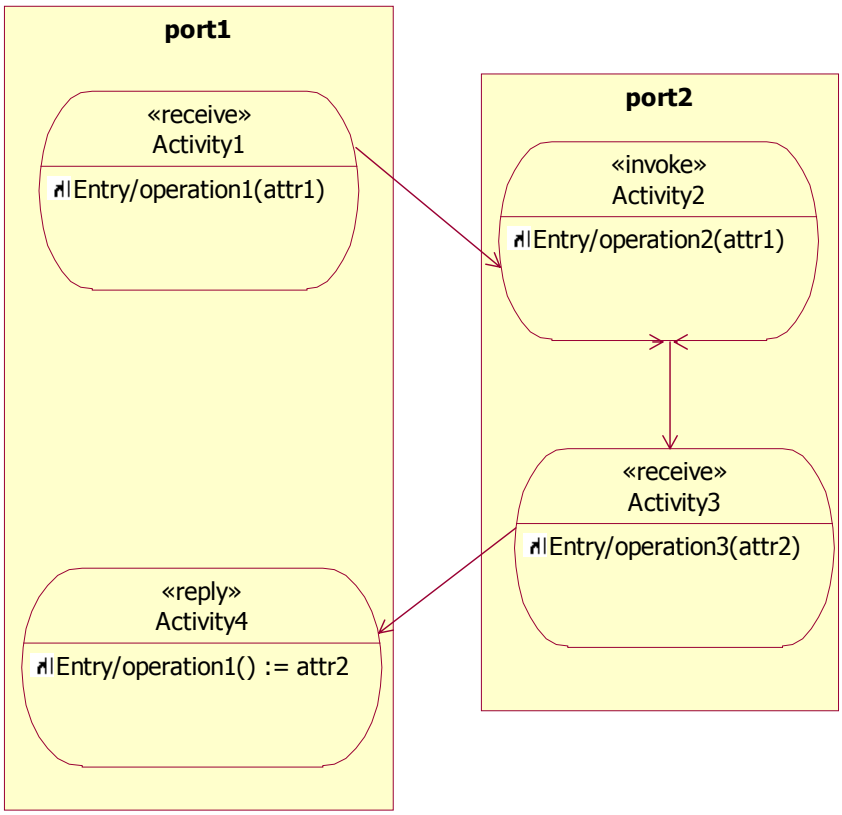


Figure 49: Receive notation

In addition to returning a response to a successful invocation, it is also possible to return an exception response. The BPEL-style of returning an exception to a caller through a reply activity is used. As in a non-exception reply statement, the left-hand side identifies the operation that has resulted in the exception. The right-hand side of the reply statement identifies the name of the exception that has been generated and the attribute that holds any exception data. The right-hand side takes the form: `exceptionName(attributeName)`. The notation for returning an exception reply is shown in Figure 50.

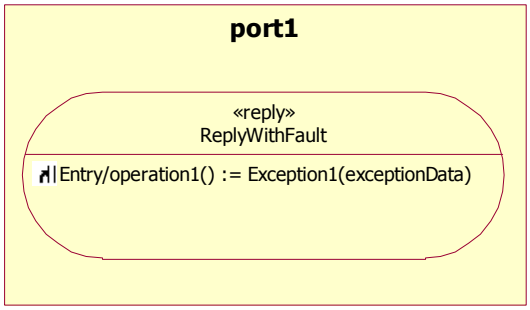


Figure 50: Reply with exception notation

MAPPING TO BPEL

Receive and reply activities are mapped to BPEL receive and reply activities with the expressions of the entry actions providing the detail in a similar manner to the mapping for invoke activities.

The BPEL generated for the receive/reply pair in Figure 47 is shown in **Error! Not a valid link.**.

```
        <receive variable="request" createInstance="yes"
            name="receive1" operation="approve" partner="customer"
            portType="LoanApprover:LoanApproval" suppressJoinFailure="no">
            <source linkName="receive1_to_invokeApprover"
transitionCondition="bpws:getVariableData('request','amount') &gt;= 10000"/>
            <source linkName="receive1_to_invokeAssessor"
transitionCondition="bpws:getVariableData('request','amount') &lt; 10000"/>
        </receive>
        …
        <reply variable="approvalInfo" name="reply" operation="approve"
            partner="customer" portType="LoanApprover:LoanApproval"
suppressJoinFailure="no">
            <target linkName="assign_to_reply"/>
            <target linkName="invokeApprover_to_reply"/>
        </reply>
```

Figure 51 BPEL generated for the receive/reply pair in Figure 46

For exception replies, the exception name and exception attribute are used to populate the faultName and faultVariable attributes of the corresponding BPEL reply activity.

Signaling exceptions

CONCEPTS

An exception can be implicitly thrown when an invoked operation returns an exception. It is also possible to explicitly throw an exception using a *throw activity*.

This document uses the Web services and BPEL convention of supporting named exceptions (faults in web services terminology) which may optionally specify exception data.

The activity shown in Figure 52 depicts throwing an OutOfStock exception on discovery that an item is out of stock.

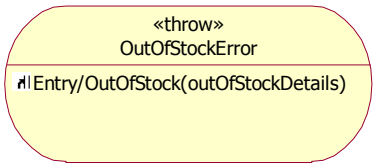


Figure 52: Throw activity

NOTATION

Throw activities, identified by the <<throw>> stereotype, have an entry action indicating the exception to be thrown. Exceptions are always thrown by name. An attribute which contains associated exception data may optionally be specified as a parameter.

Activity4 in Figure 53 is a throw activity which throws an exception named Exception1, no exception data is specified (there is no need to introduce empty parameter brackets, as in `Exception1()`, though this is also valid). Activity7 in Figure 53 is a throw activity which throws an exception names Exception2 and has data specified as attr1, which must reference an attribute of the containing process.

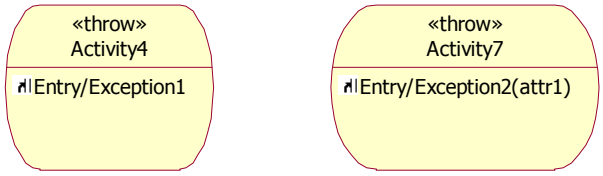


Figure 53: Throw activity notation

MAPPING TO BPEL

A <<throw>> activity is mapped to a BPEL throw activity. The entry action expression provides the faultName and the faultVariable attribute (if any).

Figure 54 shows the BPEL corresponding to the <<throw>> activity in Figure 52. The OutOfStock fault name is defined in an imported namespace which is referred to with the prefix FLT.

```
<throw name=”OutOfStockError”
       faultName="FLT:OutOfStock"
       faultVariable=”outOfStockDetails”/>
```

Figure 54: BPEL for throw activity

Terminating the process instance

CONCEPTS

A process terminates normally when all final activities (those with no outgoing control links) have completed.

A process can be explicitly terminated by introducing a control link to a final state. When a final state is reached, all activities are terminated.

For example, the receipt of a ‘cancel’ message could terminate an insurance request process during the cool off period through a control link to a Cancelled terminate activity, as shown in Figure 55.

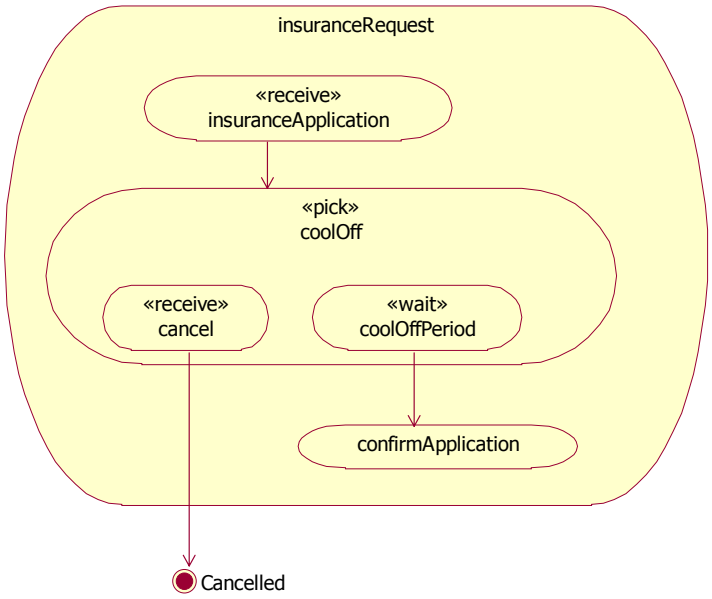


Figure 55: Explicit terminate

NOTATION

A terminate activity is shown by a 'bullseye' ( ). Figure 56 shows a final state that terminates the process on completion of Activity2.

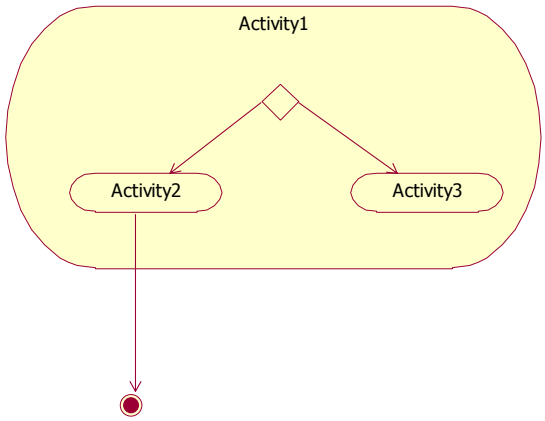


Figure 56: Explicit process termination

MAPPING TO BPEL

A final state in UML is mapped to a BPEL terminate activity. The name of the final state (if any) maps to the name of the terminate activity. Figure 57 shows the BPEL for the Cancelled final state in Figure 55.

```
<terminate name="Cancelled"/>
```

Figure 57: BPEL terminate example

Waiting

Concepts

A business process may need to wait until a particular deadline is reached before proceeding. This is modeled using a wait activity.

Figure 58 shows a wait activity that waits until a specific time and on Christmas Eve 2002, when the waitUntilChristmas activity completes, the seasonalGreeting activity runs.

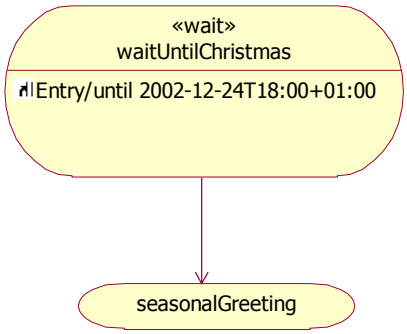


Figure 58: Example of wait activity

Notation

This is achieved using an activity stereotyped as <<wait>> that has a single entry action containing a time expression (as defined in section 0 on page 75) prefixed by either `for` or `until` to indicate a duration or deadline expression respectively.

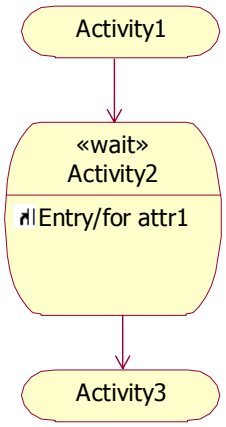


Figure 59: Wait notation

Mapping to BPEL

A <<wait>> activity is mapped to a BPEL wait activity. The entry action expression provides the time expression and whether it is placed into a for or an until attribute within the wait activity.

The BPEL for the waitUntilChristmas <<wait>> activity in Figure 58 is shown in Figure 60.

```
<wait until="2002-12-24T18:00+01:00"/>
```

Figure 60: Wait activity BPEL

Activity coordination

Basic control flow

CONCEPTS

Control dependencies between activities are modeled using *control links*. Processes support concurrent flows of control. An activity is executed when all of its incoming control links are enabled. When an activity completes, all of its outgoing control links are enabled.

Figure 61 repeats the purchase order example of section 0, which shows the use of control links to sequence activities.

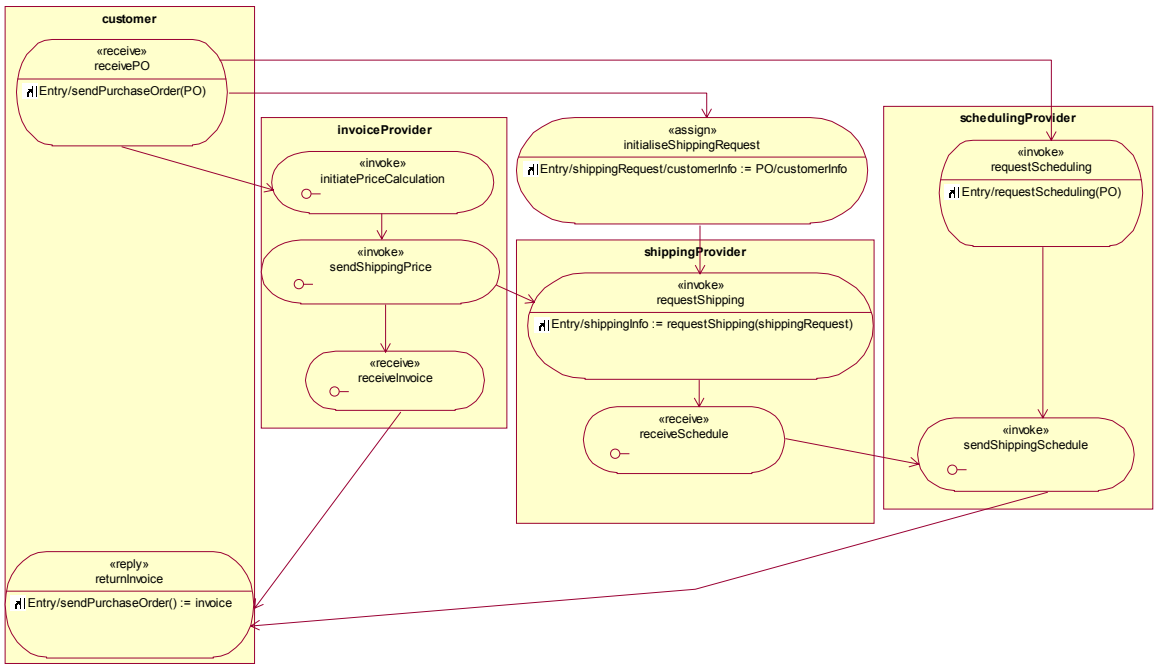


Figure 61: Use of control links

Optionally, control links may have guards. A control link with a guard is only enabled if the guard (a Boolean expression) evaluates to true when its source activity completes.

Hierarchical structure can also be used to model control flow. When a parent activity is started, any child activities without incoming control links are started. When all child activities that do not have outgoing control links have completed then the parent activity completes.

NOTATION

A control link is shown by an arrow connecting two activity nodes.

An activity node may have zero, one, or more incoming and outgoing control links

An activity node may be nested inside another activity node. Control links may cross activity boundaries; that is, a nested activity can have an incoming or outgoing control link to a node outside the node in which it is nested.

Loops in control structure are not permitted. The structured <<loop>> construct introduced in Section 0 should be used.

Figure 62 demonstrates the notation for control flow with nesting.

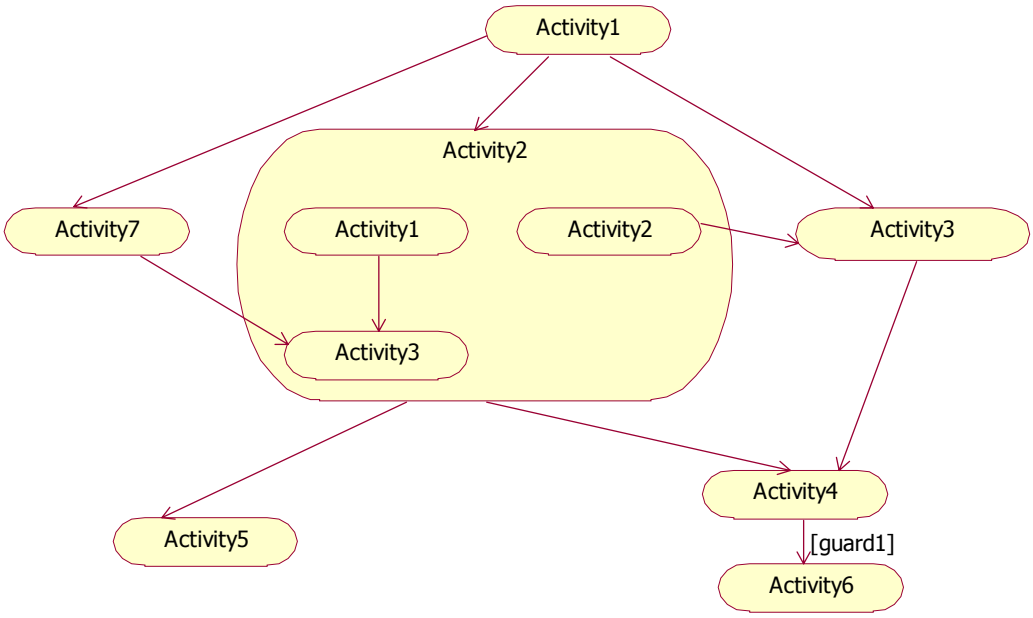


Figure 62 Notation for control links and nesting

MAPPING TO BPEL

Mapping complex control structure to the structured constructs of BPEL is not straightforward if you want to obtain BPEL that is close to 'hand-written' BPEL as possible.

Where there is a set of activities (either at the top level or within a composite activity) that are executed one after another, one at a time, then a BPEL sequence activity is generated to contain those activities.

For a more complex control structure a BPEL flow can be used. All activities could be placed into a BPEL flow with the appropriate links. This would be semantically correct, but a BPEL programmer would be more likely to structure the flow using sequences.

Therefore the following heuristic is used to structure the flow. If all activities with a common direct parent that involve interaction with the same partner can be placed in a sequence, the activities are placed in the sequence with any non-interaction activities (assign, wait, and so on) that have control links only to other elements within the sequence. (Note that it is not possible to have multiple concurrent activities interacting with the same partner.) Any elements that are not placed within a sequence become top-level activities within the flow.

The example in Figure 61 maps to the BPEL shown in Figure 63 (detail omitted).

```
<flow name="PurchaseOrder">
   <sequence name="customerInteraction">
   …
```

```
   </sequence>
   <sequence name="invoiceProviderInteraction">
   …
   </sequence>
   <assign name="initialiseShippingInformation">
   …
   </assign>
   <sequence name="shippingProviderInteraction">
   …
   </sequence>
   <sequence name="schedulingProviderInteraction">
   …
   </sequence>
</flow>
```

Figure 63: BPEL control flow example

The UML modeler also has the option of using hierarchical nesting when appropriate, however flows and sequences are not distinguished in UML. Each level in the hierarchy is recursively given the same treatment as described above.

Decision and merge

CONCEPTS

Decision and merge nodes can be introduced to fine-tune control of concurrent behavior.

Decisions enable the behavior of a process to be dependent on its current state. When control reaches a decision node, the guards on each of its outgoing control links are evaluated, when a guard evaluates to true that control link is enabled and there is no further evaluation of guards.

At most one control link can have the guard 'otherwise' indicating that that control link is enabled if the guard for all the other control links evaluate to false.

A merge node allows any one of a number of incoming control links to cause an outgoing control link to be enabled. A merge node can be used to merge the alternative flows of control created by a decision node to indicate that behavior after the merge is common in all cases.

Figure 64 shows the use of a decision node and a merge node to provide different request handling behavior depending on whether the customer making the request has gold status or not.

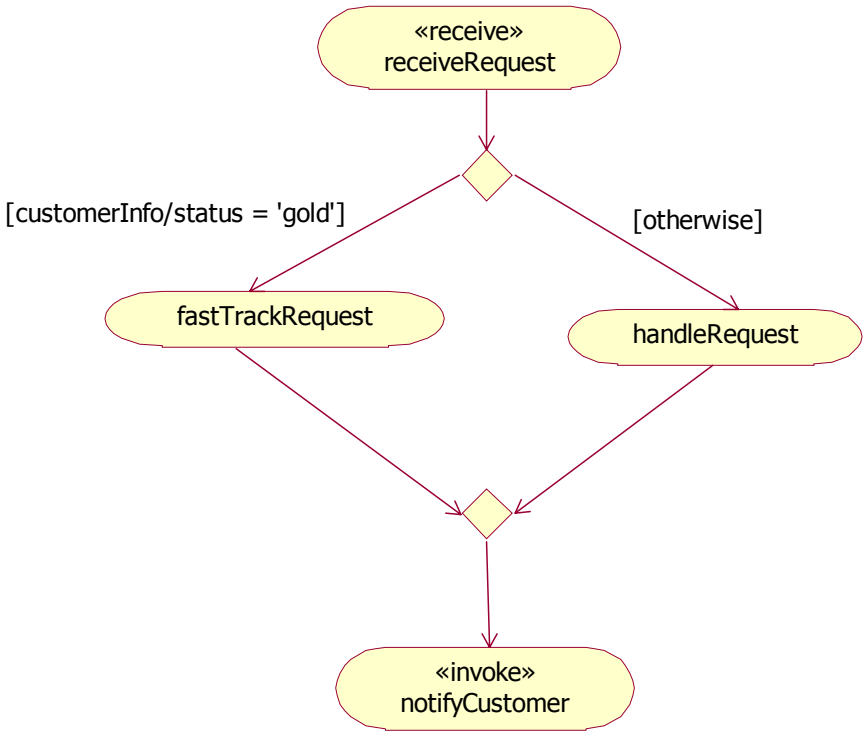


Figure 64: Example of decision and merge nodes

NOTATION

The notation is illustrated by Figure 65. Decision and merge nodes are both shown as diamonds (there is one of each in the diagram).

Decision nodes are choices, where each outgoing control link is guarded by an expression. The guard is shown within square brackets in a label on the control link. Optionally one (and only one) outgoing control can be guarded by the expression 'otherwise'.

Merge nodes have a series of incoming control links and one outgoing control link.

Note that if you draw a 'diamond' node with multiple incoming and outgoing links then this means a merge followed by a condition. That is, the node will be evaluated when any of the incoming links is activated and only one of the outgoing links is activated depending on the guards on outgoing links.

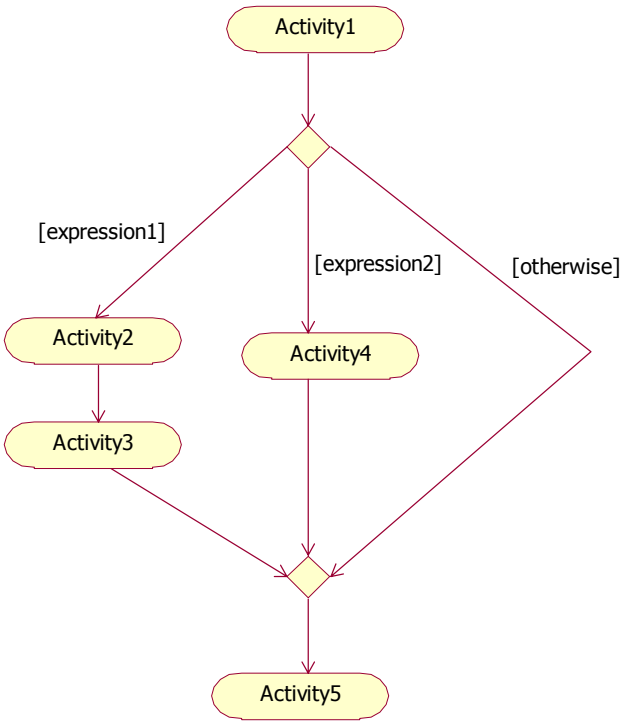


Figure 65 Notation for decision and merge nodes

In UML a decision node can have only one incoming link and a merge can have only one outgoing link. If a diamond is drawn with multiple incoming and outgoing links, then it is interpreted as a merge node followed by a decision node.

This means that implicit fork and join for control links is not available for decision and merge nodes. To perform a join before a decision node, or a fork after a merge node, explicit join or fork nodes must be introduced.

The Marketplace process shown in Figure 66 demonstrates the use of a join (solid bar) before a decision and a fork (also represented by a solid bar) after a merge.

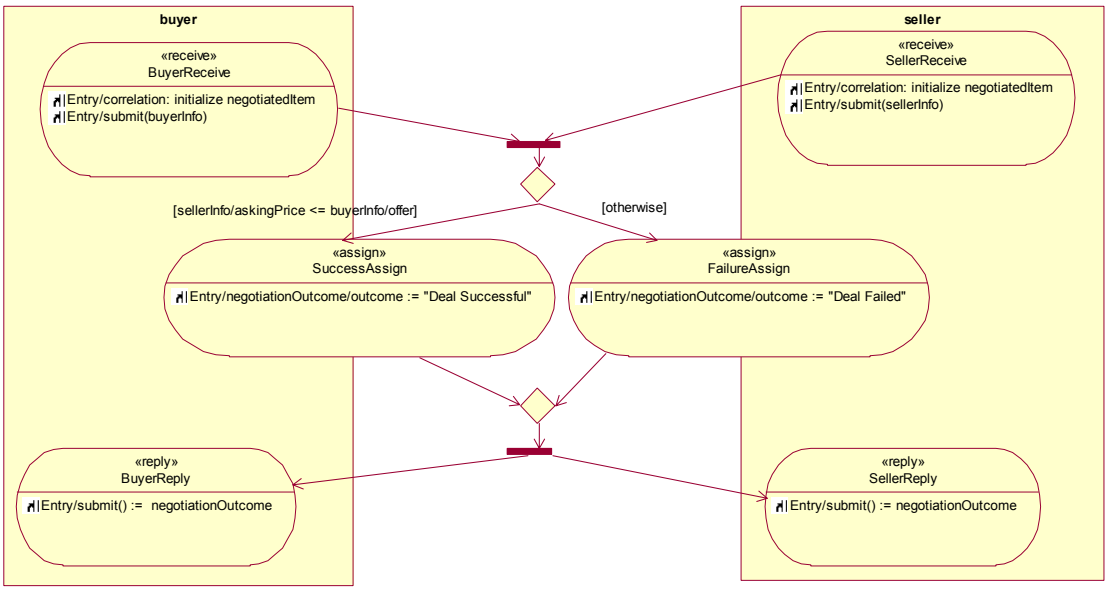


Figure 66: Use of explicit fork and join nodes

STYLE GUIDELINES

Decision node behavior can be achieved through placing guards on multiple outgoing transitions from an activity.

However, you are recommended to introduce a decision node for clarity whenever a mutually-exclusive choice is made (as it is in this case). When using guarded transitions, there can be multiple transitions that evaluate to true and they would all be activated.

MAPPING TO BPEL

A decision node and corresponding merge (the end of the process is deemed to be an implicit merge) is mapped to a BPEL switch statement.

Figure 64 shows a well-nested decision and merge node and maps to the BPEL switch activity outlined in Figure 67.

```
<switch>
   <case condition="bpws:getVariableData('customerInfo', 'status') = 'gold'">
      <!--- fast track request --->
   </case>
   <otherwise>
      <!--- handle request --->
   </otherwise>
</switch>
```

Figure 67: BPEL switch activity

Looping

CONCEPTS

A business process can repeat a particular activity while a condition evaluates to true. Rather than permitting unstructured loops in activity graphs this profile supports an explicit structured looping node. This conforms to BPEL and other workflow languages.

The loop activity provides support for while loops as in BPEL; if the guard condition is false on entry to the loop, then the loop body is never executed (contrast with a do-until loop, which always performs the loop body at least once).

Figure 68 shows a loop activity that repeatedly executes the DoSomething activity while the numberOfItems attribute of the orderDetails attribute is greater than 100.

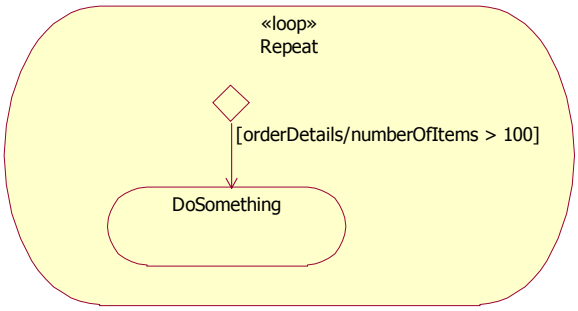


Figure 68: Loop example

NOTATION

A looping node is shown as an activity with the stereotype <<loop>>, which contains a decision node and an activity to be repeated, with a control link from the decision node to the activity. The guard on the control link provides the Boolean expression which determines whether the activity is executed each time round the loop.

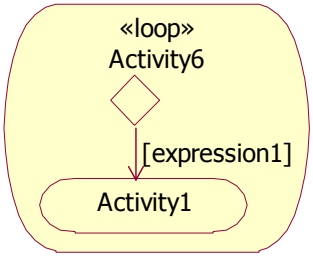


Figure 69 Notation for loop activities

MAPPING TO BPEL

A <<loop>> activity maps to a BPEL while activity. Figure 70 shows the BPEL corresponding to the loop activity in Figure 68.

```
<while condition=
           "bpws:getVariableData(orderDetails,numberOfItems) > 100">
  <scope name="DoSomething">
    ...
  </scope>
</while>
```

Figure 70: BPEL while activity example

Pick

CONCEPTS

It is often useful to group a set of receive activities and optionally a wait activity so that when one of them occurs, the remainder are disabled. For example, after sending an approval request to an approver, an order process could wait for either an accept message, a reject message, or a deadline to be reached – whichever happens first.

BPEL refers to this notion as pick activity, and this profile also uses the term pick.

Figure 71 shows the use of a pick within an order process. This pick might be placed within a loop to enable the order process to respond to a series of inputLineItem requests followed by an orderComplete request, with inactivity triggering timeout handling. When control reaches the pick node (a decision node stereotyped with <<pick>>) the ReceiveLineItem, ReceiveCompletion and OrderCompletionTimeout is enabled; when one of the activities takes place the others are disabled. ReceiveLineItem and ReceiveCompletion are both receive activities that are triggered by receipt of a request received through the buyer port. OrderCompletionTimeout is a wait activity which waits for three days and ten hours.

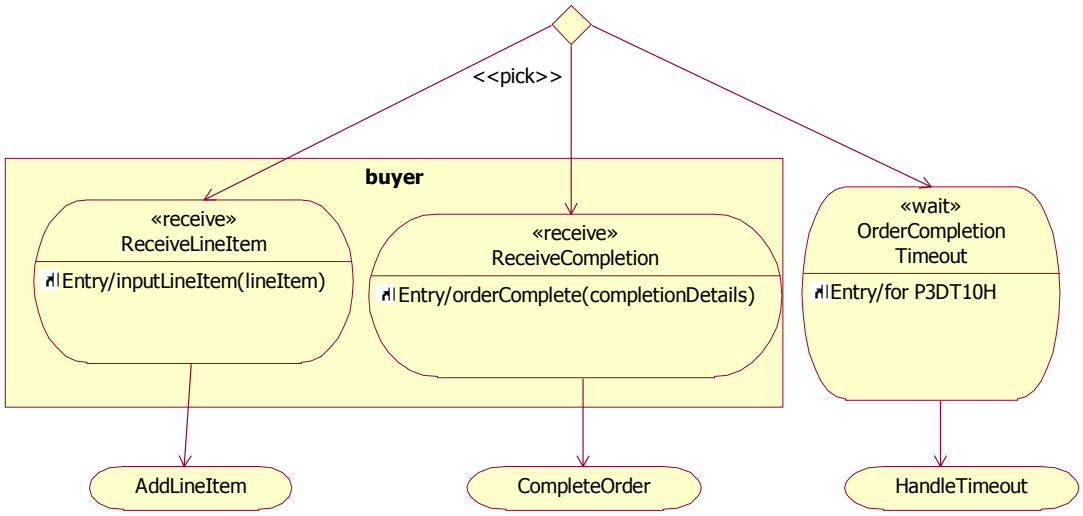


Figure 71: Pick example

NOTATION

Figure 72 shows the notation for a pick activity, which is shown as a decision node with the stereotype <<pick>>. The outgoing control links from the decision node must all connect to activity nodes stereotyped by <<receive>> or <<wait>>.

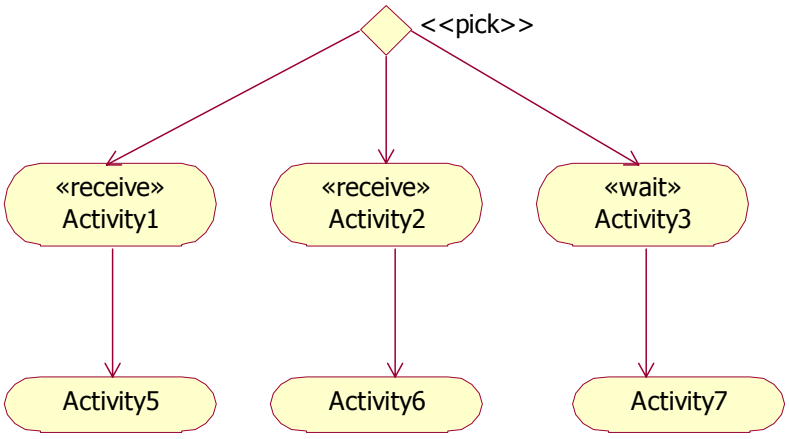


Figure 72: Pick notation

The profile also supports the alternative notation shown in Figure 73, which is more in line with UML 2, where interruptible regions can be used to represent pick behavior. However,

UML 2 uses nesting and therefore does not permit graphical partitions to be used to represent partners (see Section 0 for the alternate partition notation that is used in this case).

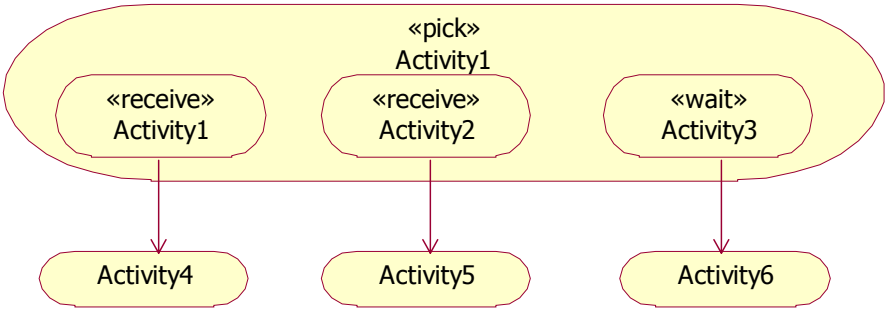


Figure 73: Pick alternative notation

Error handling

Exception handling

CONCEPTS

When an invoked operation throws an exception, or a throw activity explicitly throws an exception, normal execution within the containing scope is terminated.

An exception can be caught within the containing scope so that error recovery behavior can be performed.

Exception handling in this profile follows the BPEL fault handling semantics, whereby exceptions can be thrown and caught by name or type.

Figure 74 shows a catch activity at the top level of a process which catches any loanProcessFault that occurs in the process and places it in the error attribute of the process. The process responds to a synchronous invocation of its approve operation, so the recovery behavior is to return the exception that occurred. The notation for returning an exceptional response is the same as that for returning a normal response, except that the right-hand-side of the expression contains a fault name and the name of the attribute containing the exception data[14].

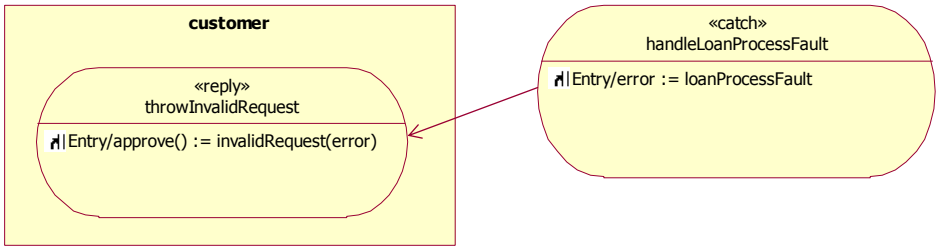


Figure 74: Catch activity

NOTATION

[14] The approach of using a reply activity to return a fault is consistent with BPEL.

One or more catch activities can be placed within a parent activity to catch exceptions that occur in any of its children. A catch activity is scoped by its parent and catches matching exceptions that occur in any child of its parent.

Each catch activity has an entry action which specifies the kind of exception that it can handle and the attribute to which the exception should be copied when caught. The exception name 'Any' is used to indicate that all exceptions should be caught.

Example, to catch activities within a process:

- `attr1 := faultName1` – catches a *faultName1* exception and copies the exception into *attr1*
- `attr1` – where *attr1* is the name of an attribute of the process, catches an exception of a type matching that of *attr1* and copies the exception into *attr1*
- `faultName1` – where *faultName1* is not the name of an attribute of the process, catches a *faultName1* exception (any associated exception data is lost).
- `Any` – catches any exception not caught by a more specific catch activity.

Outgoing control links from a catch activity indicate the behavior to be performed on catching the exception. The recovery behavior can be nested within the catch activity if using hierarchical nesting to indicate control flow.

Figure 75 shows two catch activities that handle exceptions that occur within the scope of Activity1. Activity6 catches exceptions of type Exception1 and performs Activity7 followed by Activity8 as recovery behavior. Activity9 catches any other exception that can occur within the scope of Activity1 and performs Activity10 as recovery behavior.

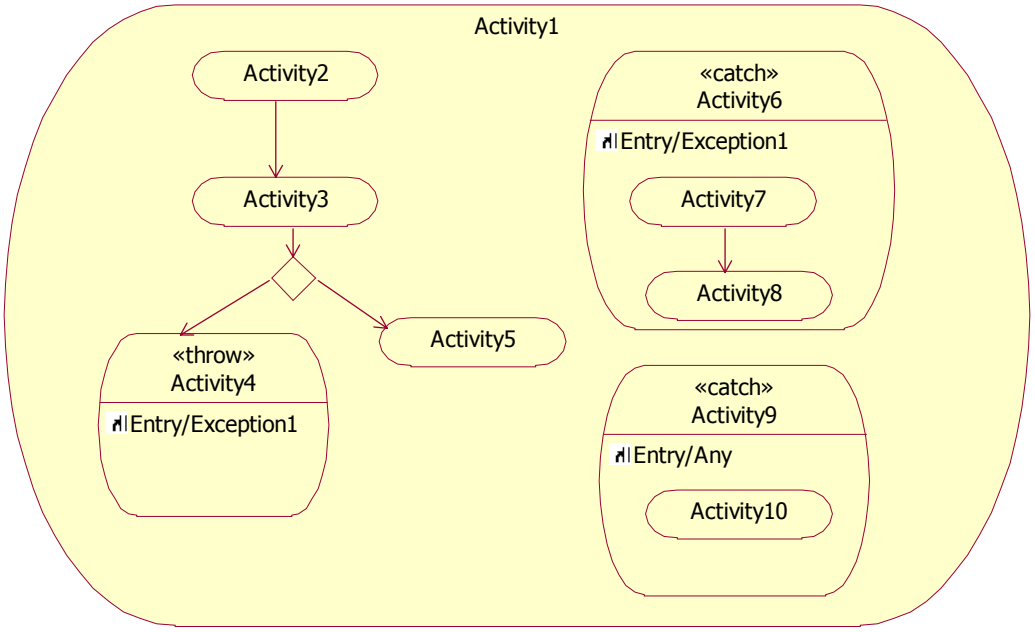


Figure 75: Catch notation

When handling exceptions that occur as a result of an invoke it is necessary to specify a variable into which the exception object should be placed if that exception object needs to be accessed. The type of this attribute determines the exceptions that may be caught.

STYLE GUIDELINES

Scopes without any incoming or outgoing control links can be introduced purely for the scoping of exception and compensation handlers (compensation is introduced in Section 0). Such activities can be omitted from activity diagrams that do not show exception handling. The handling of exceptions within a scope can then be shown on a separate diagram.

MAPPING TO BPEL

Catch activities at the top level map to corresponding catch activities within the BPEL faultHandlers element at the process scope.

Nested catch activities cause a BPEL scope to be introduced corresponding to the parent activity. Catch activities then map to catch activities within the faultHandlers element of that scope.

The catch activity in Figure 74 is at the top-level and therefore maps to the process-level fault handler shown in Figure 76.

```
<process>
    <faultHandlers>
        <catch faultVariable="error"
faultName="LoanAssessor:loanProcessFault">
            <reply variable="error" faultName="invalidRequest"
                name="throwInvalidRequest" operation="approve"
                partner="customer" portType="LoanApprover:LoanApproval"
suppressJoinFailure="no"/>
        </catch>
    </faultHandlers>
    ...
</process>
```

Figure 76: BPEL top-level fault handler

Compensation

CONCEPTS

In some cases it can be necessary to reverse work that has already been performed (for example, due to a failure of a subsequent piece of work).

Compensation handler activities can be defined to reverse the work performed by a scope, if necessary.

Compensation handler activities are not executed when control reaches the parent activity. If the parent activity completes successfully then the compensation handler is installed. Compensation can be triggered by a *compensate* activity.

We follow the BPEL semantics for compensation and when it can be triggered. In particular, a compensate activity is only permitted in the following places:

- In a `catch activity` of the scope that immediately encloses the scope for which compensation is to be performed.
- In the `compensation` handler of the scope that immediately encloses the scope for which compensation is to be performed.

Comment: Include an example to illustrate concepts

Compensation examples are currently under development and will be added when available.

NOTATION

A compensation activity has the stereotype <<compensationHandler>>. At most one compensation activity can be placed within an activity to compensate the work performed by the children of that activity.

A compensate activity has the stereotype <<compensate>>.

Figure 77 shows an activity X which contains an activity doX that is executed when control reaches X. On successful completion of X, the compensationHandler Activity3 is installed.

If an exception of type Exception1 is thrown within the scope of Activity2 (which includes activity Y) then catch activity Activity2 executes, causing Activity1 to compensate the scope X.

Compensating scope X causes the causes Activity3 to execute (if installed) which causes the reserveX activity to take place.

Activities to be performed on compensation can be nested within the compensationHandler activity (as shown) or a control link from the compensationHandler activity to the compensation activity may be used.

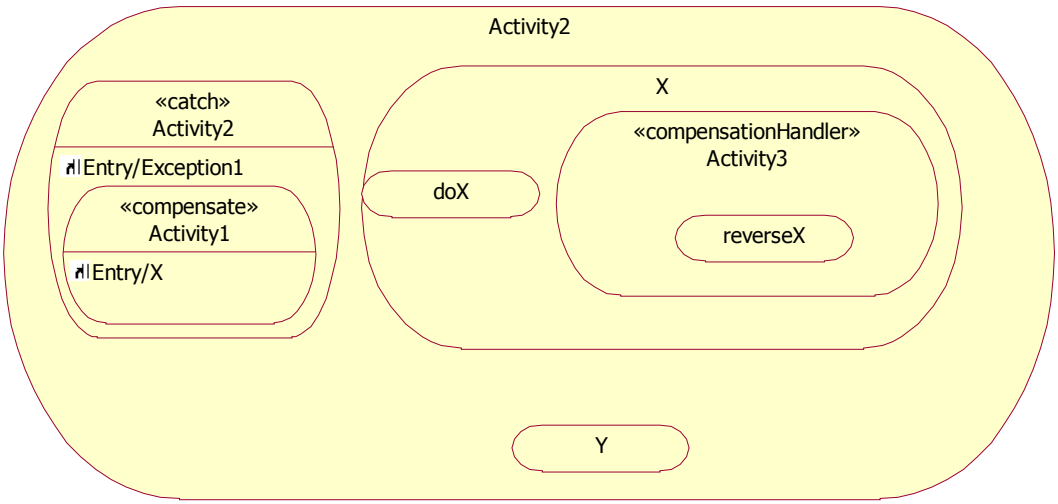


Figure 77: Compensation notation

MAPPING TO BPEL

To be added when available.

Comment: Add some text to this section!

Example – Marketplace

This example is based on a sample provided with the BPWS4J runtime for BPEL [].

The marketplace process mediates a negotiation between a buyer and a seller. When the marketplace process has an asking price from the seller and an offer from the buyer (which can happen in either order) it checks to see if the offer meets or exceeds the asking price. A response indicating the success or failure of the negotiation is returned to both the buyer and the seller.

In this case the marketplace process offers Buyer and Seller interfaces to its partners and does not use any interfaces provided by its partners. The Buyer interface introduces an operation for submitting an offer, and the Seller interface introduces an operation for submitting an asking price. A class, stereotyped <<messageContent>>, is introduced for each parameter type used by the operations in these interfaces. Note that messageContent rather than data is used here to indicate that the grouping of attributes is for the purposes of a message exchange rather than to introduce a new reusable data type.

The dependency arrows indicate which classes are used by which interfaces. Dependency arrows are optional and are introduced here only for clarity.

The interface and class definitions are grouped together in a BuyerSeller package.

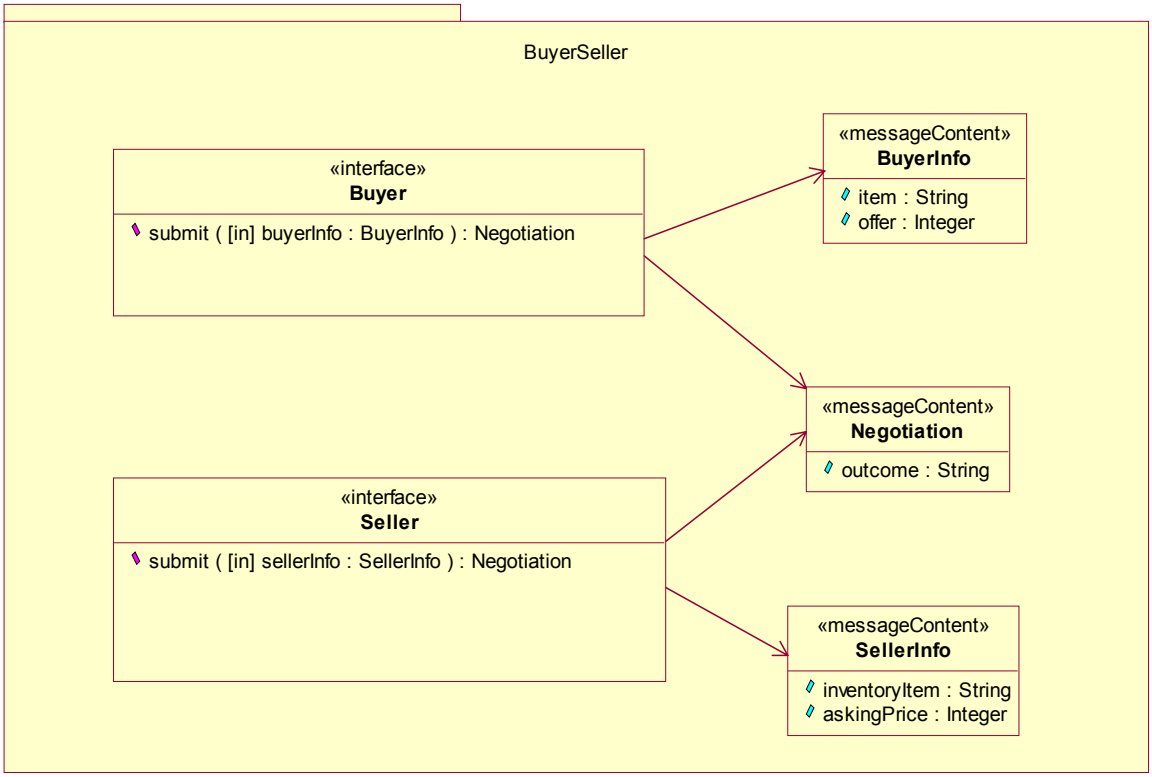


Figure 78 – Interfaces

The Marketplace example uses the identifier of the negotiated item for correlation. Both the SellerInfo and BuyerInfo classes need to carry this correlation information. In the case of the BuyerInfo class the item attribute is used, in the case of the SellerInfo class the inventoryItem attribute is used.

Therefore, a negotiatedItem property, with aliases for both BuyerInfo and SellerInfo, is introduced. The property and aliases are placed in a Properties class within the BuyerSeller package as shown in Figure 79. The query expressions for extracting the properties from the messages do not appear on the class diagram since they are placed in the defaultExpression of the return parameter. For SellerInfo the expression is /inventoryItem, and for BuyerInfo it is /item.

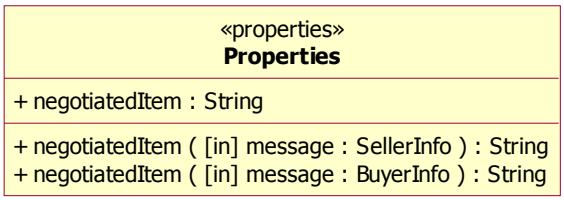


Figure 79: BuyerSeller properties

The Marketplace process participates in the two protocols shown in Figure 80.

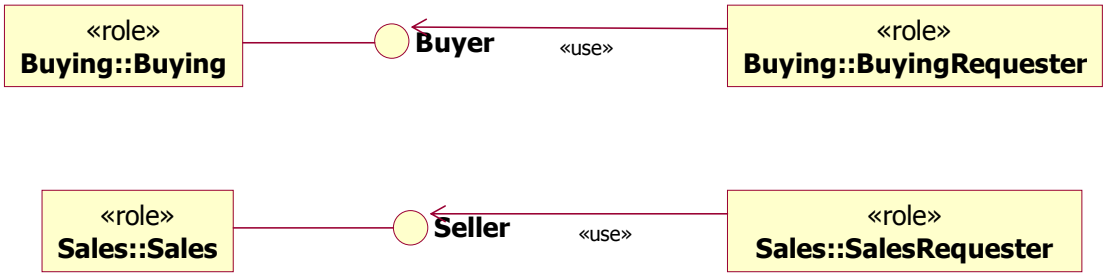


Figure 80: Marketplace protocols

Next, the Marketplace business process is defined. The process has two ports through which it provides the Buyer and Seller interfaces. The Marketplace process uses three pieces of data: buyerInfo, sellerInfo and negotiation. A correlation set containing the negotiatedItem property is introduced.

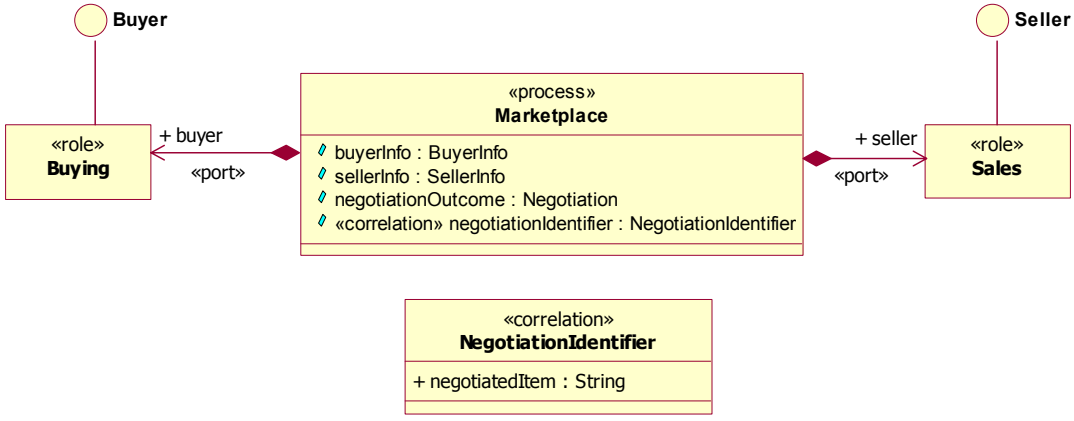


Figure 81: Marketplace process, ports, attributes and correlation

Finally, the behavior of the Marketplace business process is defined. This is done by attaching an activity graph to the Marketplace class. A partition corresponding to each port

is defined. Many activities within an automated business process represent an interaction through a port. The port is indicated by placing the activity in the appropriate partition.

In the Marketplace example, either the buyer or the seller can initiate a process by sending a submit request referring to an item for which there is not an existing process instance. If the buyer sends the initiating request for a particular item then a subsequent message from the seller containing matching correlation data is received by the same process instance.

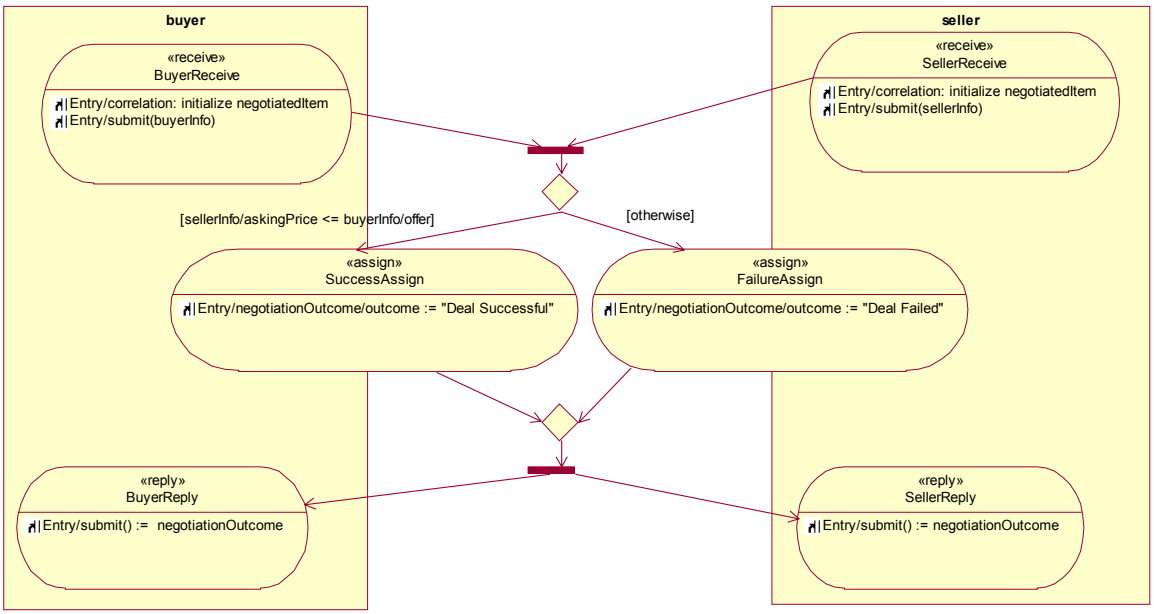


Figure 82: Behavior of the Marketplace process

Example – Loan approval

This loan approval example is based on the loan approval sample provided with BPEL4J, which is in turn based on an example from the BPEL specification.

The loan approval process is initiated when a customer makes a loan request. Approval from an approver partner must be received if the loan is of high value, or if a risk assessment from an assessment does not determine the loan to be low risk. A synchronous reply indicating whether or not the approval has succeeded is returned to the customer.

The packaging structure of the loan approval example is shown in Figure 83. The LoanApprover and LoanAssessor packages contain the interfaces that are used in the example and the classes referred to by those interfaces. The LoanDefinitions package contains classes that are referred to in both the LoanApprover and LoanAssessor packages. The protocols through which the loan approval process communicates with its partners are defined in the LoanApprovalProtocols package. The loan approval process is defined in the LoanApprovalProcess package.

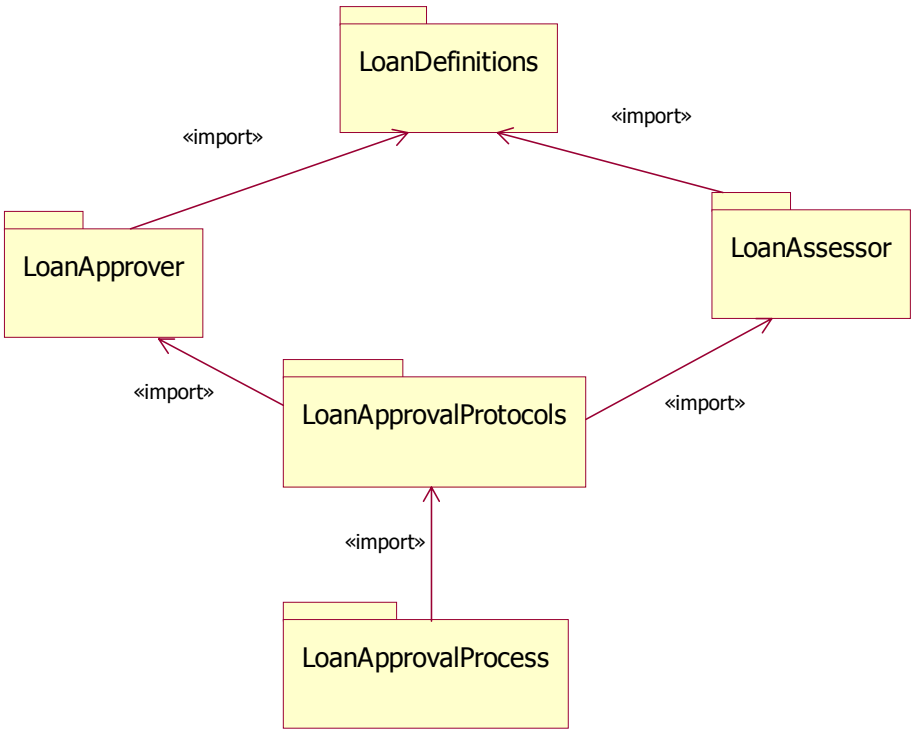


Figure 83: Loan approval packing

The classes introduced in the LoanDefinitions package are shown in Figure 84. These are stereotyped by <<messageContent>> to indicate that they are used to introduce a message type and should lead to the generation of WSDL message types and not XSD types.

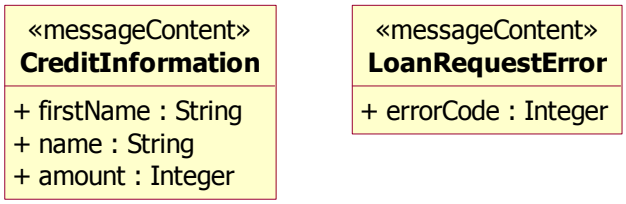


Figure 84: LoanDefinitions package

The LoanApproval package, shown in Figure 85, introduces an interface for requesting a loan approval and receiving a synchronous response.

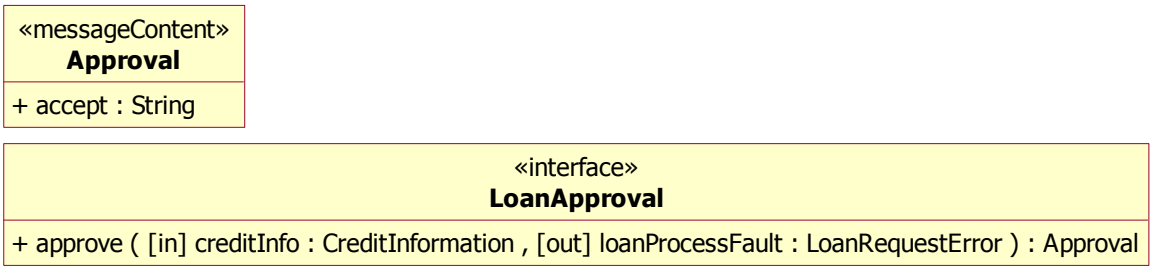


Figure 85: LoanApproval package

Figure 86 shows the LoanAssessment package which introdces an interface for requesting a risk assessment (and receiving a synchronous response).

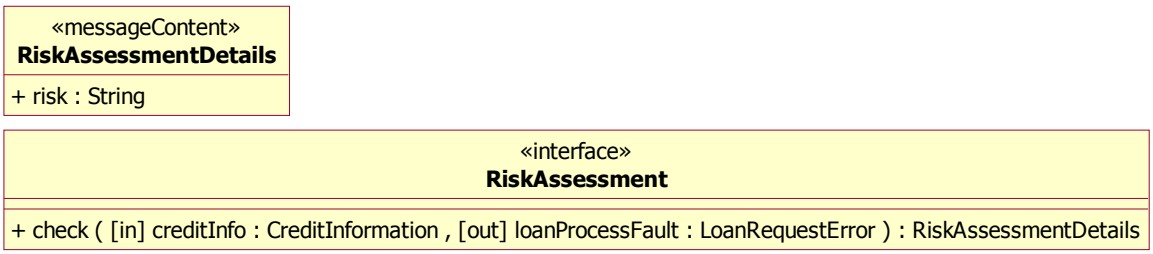


Figure 86: LoanAssessment package

The LoanApprovalProtocols package, shown in Figure 87, introduces the collaborations in which the loan approval process participates. Each protocol is placed in a subpackage of LoanApprovalProtocols, the LoanApproval:: and RiskAssessment:: prefixes to the role names indicate the subpackage to which they belong.

Note that although the process has three partners, only two protocols are provided. This is because the process participates in the LoanApproval protocol twice, once as an Approver (to the customer) and once as an ApprovalRequester (to the approver).

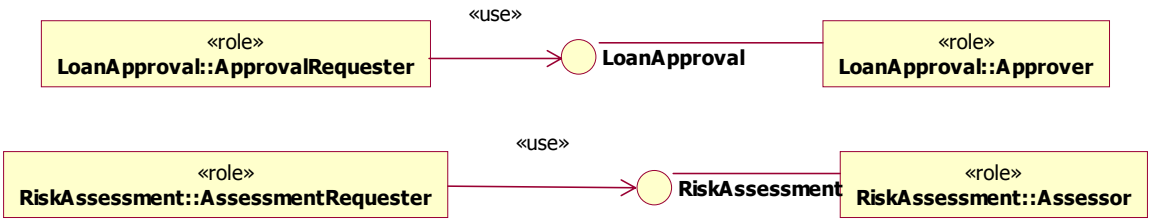


Figure 87: LoanApprovalProtocols package

The loan approval process is defined in the LoanApprovalProcess as shown in Figure 88. The loan approval process has four attributes for storing the state of the process, the types of these attributes are classes that are defined in the packages described earlier.

LoanApproval participates in collaborations with three partners and therefore has three ports: customer, assessor and approver.

The interfaces provided to, or required from, the customer are defined by the Approver role, in this case, the LoanApproval process provides the LoanApproval interface through the customer port.

The interfaces provided to, or required from, the approver are defined by the ApprovalRequester role, in this case the process requires the LoanApproval interface through the approver port.

The interfaces provided to/required from the assessor are defined by the AssessmentRequester role, in the case the process requires the RiskAssessment interface via the approver port.

There is no need for correlation in this example since there are no messages initiated by partners after the initial request that creates a new instance of the process.

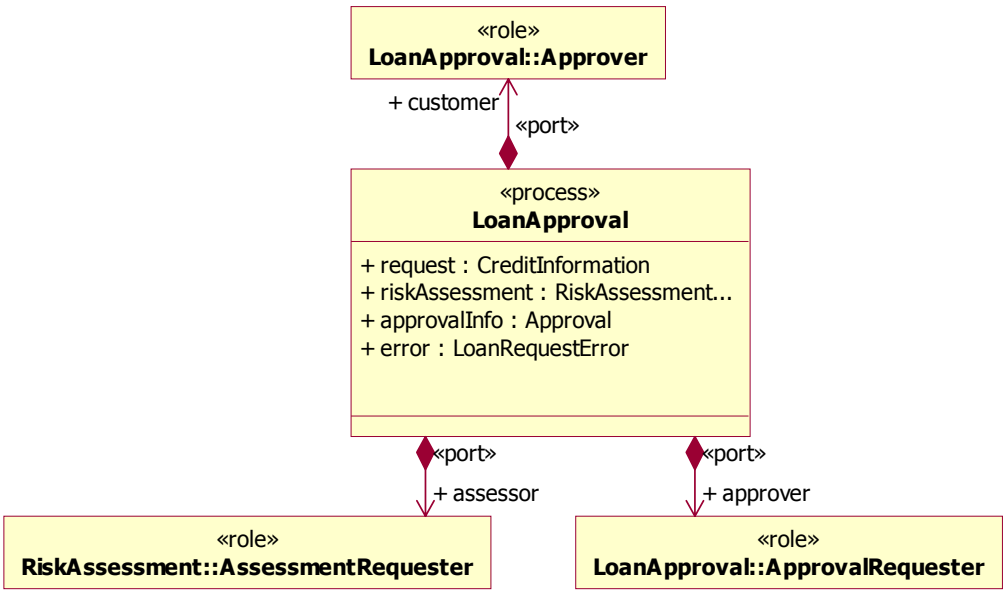


Figure 88: LoanApprovalProcess package

The behavior of the process is shown in Figure 89. The process starts when an approve request is received from the customer. The amount and risk-level (obtained from the assessor, if necessary) of the loan determine whether an approval is required from the approver.

Figure 89: Activity graph for LoanApproval process

The merge node before the invokeApprover activity indicates that there are two paths that can lead to the execution of this activity (either the requested loan amount is over 10,000 or it is below that amount but not low risk). The merge node is required since the meaning of two incoming control links to the same activity is that both must be activated for the activity to execute.

The LoanApproval process invokes operations which may throw exceptions. Exception handling is not shown in Figure 89; instead we show error handling on a separate diagram.

Figure 90 shows a catch activity, handleLoanProcessFault, which is defined at the top-level of the process and therefore applies to any activity in the process. If a loanProcessFault is thrown then it is caught by the handleLoanProcessFault activity and placed into the error attribute of the process. It is then returned as an invalidRequest exception from the approve operation which was synchronously invoked by the customer.

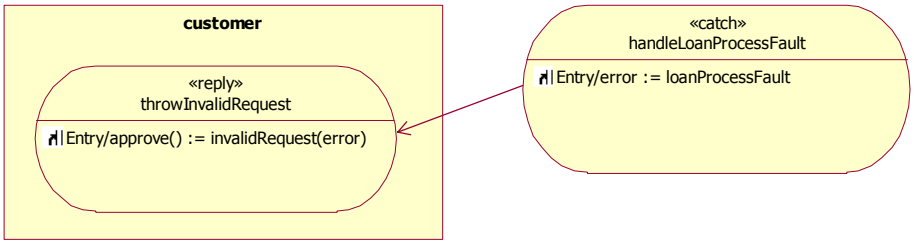


Figure 90: Exception handling for loan approval process

Loan approval WSDL

WSDL for the LoanDefinitions package:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    targetNamespace="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanDefinitions/"
    xmlns:LoanDefinitions="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanDefinitions/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <wsdl:message name="CreditInformation">
        <wsdl:part name="firstName" type="xsd:string"/>
        <wsdl:part name="name" type="xsd:string"/>
        <wsdl:part name="amount" type="xsd:int"/>
    </wsdl:message>
    <wsdl:message name="LoanRequestError">
        <wsdl:part name="errorCode" type="xsd:int"/>
    </wsdl:message>
</wsdl:definitions>
```

WSDL for LoanApprover package:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    targetNamespace="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanApprover/"
    xmlns:LoanApprover="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanApprover/"
    xmlns:LoanDefinitions="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanDefinitions/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <wsdl:import
   location="http://localhost:8080/BPEL/LoanApproval/LoanDefinitions.wsdl"
namespace="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanDefinitions/"/>
    <wsdl:message name="Approval">
        <wsdl:part name="accept" type="xsd:string"/>
    </wsdl:message>
```

```
    <wsdl:portType name="LoanApproval">
        <wsdl:operation name="approve">
            <wsdl:input message="LoanDefinitions:CreditInformation"
name="creditInfo"/>
            <wsdl:output message="LoanApprover:Approval"/>
            <wsdl:fault message="LoanDefinitions:LoanRequestError"
name="loanProcessFault"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

WSDL for LoanAssessor package:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    targetNamespace="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanAssessor/"
    xmlns:LoanAssessor="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanAssessor/"
    xmlns:LoanDefinitions="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanDefinitions/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <wsdl:import

location="http://localhost:8080/BPEL/LoanApproval/LoanDefinitions.wsdl"
namespace="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanDefinitions/"/>
    <wsdl:message name="RiskAssessmentDetails">
        <wsdl:part name="risk" type="xsd:string"/>
    </wsdl:message>
    <wsdl:portType name="RiskAssessment">
        <wsdl:operation name="check">
            <wsdl:input message="LoanDefinitions:CreditInformation"
name="creditInfo"/>
            <wsdl:output message="LoanAssessor:RiskAssessmentDetails"/>
            <wsdl:fault message="LoanDefinitions:LoanRequestError"
name="loanProcessFault"/>
        </wsdl:operation>
```

```
    </wsdl:portType>
</wsdl:definitions>
```

WSDL, with BPEL extensions, for LoanApprovalProtocols package:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    targetNamespace="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanApprovalProtocols/"
    xmlns:LoanApprovalProtocols="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanApprovalProtocols/"
    xmlns:LoanApprover="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanApprover/"
    xmlns:LoanAssessor="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanAssessor/"
    xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
    xmlns:slnk="http://schemas.xmlsoap.org/ws/2003/03/service-link"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <wsdl:import
        location="http://localhost:8080/BPEL/LoanApproval/LoanApprover.wsdl"
namespace="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanApprover/"/>
    <wsdl:import
        location="http://localhost:8080/BPEL/LoanApproval/LoanAssessor.wsdl"
namespace="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanAssessor/"/>
    <slnk:serviceLinkType name="LoanApprovalProtocols:LoanApproval">
        <slnk:role name="Approver">
            <slnk:portType name="LoanApprover:LoanApproval"/>
        </slnk:role>
    </slnk:serviceLinkType>
    <slnk:serviceLinkType name="LoanApprovalProtocols:RiskAssessment">
        <slnk:role name="Assessor">
            <slnk:portType name="LoanAssessor:RiskAssessment"/>
        </slnk:role>
    </slnk:serviceLinkType>
</wsdl:definitions>
```

Loan approval BPEL

BPEL for LoanApproval process:

```
<?xml version="1.0" encoding="UTF-8"?>
<process abstractProcess="no" enableInstanceCompensation="no"
    name="LoanApproval" suppressJoinFailure="yes"
    targetNamespace="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanApprovalProcess/LoanApproval/"
    variableAccessSerializable="no"
    xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
    xmlns:LoanApproval="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanApprovalProcess/LoanApproval/"
    xmlns:LoanApprovalProtocols="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanApprovalProtocols/"
    xmlns:LoanApprover="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanApprover/"
    xmlns:LoanAssessor="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanAssessor/"
xmlns:LoanDefinitions="http://www.bpel-
examples.ibm.com/BPEL/LoanApproval/LoanDefinitions/">
    <partners>
        <partner name="approver" partnerRole="Approver"
serviceLinkType="LoanApprovalProtocols:LoanApproval"/>
        <partner name="assessor" partnerRole="Assessor"
serviceLinkType="LoanApprovalProtocols:RiskAssessment"/>
        <partner myRole="Approver" name="customer"
serviceLinkType="LoanApprovalProtocols:LoanApproval"/>
    </partners>
    <variables>
        <variable messageType="LoanDefinitions:CreditInformation"
name="request"/>
        <variable messageType="LoanAssessor:RiskAssessmentDetails"
name="riskAssessment"/>
        <variable messageType="LoanApprover:Approval" name="approvalInfo"/>
        <variable messageType="LoanDefinitions:LoanRequestError"
name="error"/>
    </variables>
    <faultHandlers>
```

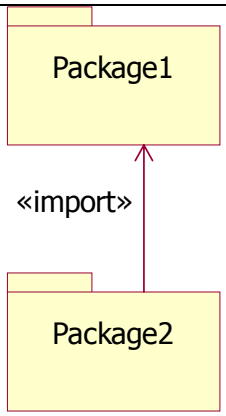
```
        <catch faultName="LoanApprover:loanProcessFault"
faultVariable="error">
            <reply faultName="invalidRequest" name="throwInvalidRequest"
                operation="approve" partner="customer"
                portType="LoanApprover:LoanApproval" variable="error"/>
        </catch>
    </faultHandlers>
    <flow>
        <links>
            <link name="receive1_to_Switch"/>
            <link name="invokeApprover_to_reply"/>
            <link name="assign_to_reply"/>
        </links>
        <flow>
            <links>
                <link name="Empty_to_invokeApprover"/>
                <link name="Empty1_to_invokeApprover"/>
            </links>
            <switch name="Switch">
                <case
condition="bpws:getVariableData(&apos;request&apos;,&apos;amount&apos;) &gt;=
10000">
                    <empty name="Empty">
                        <source linkName="Empty_to_invokeApprover"/>
                    </empty>
                </case>
                <case
condition="bpws:getVariableData(&apos;request&apos;,&apos;amount&apos;) &lt;
10000">
                    <sequence>
                        <invoke inputVariable="request"
                            name="invokeAssessor" operation="check"
                            outputVariable="riskAssessment"
                            partner="assessor"
portType="LoanAssessor:RiskAssessment"/>
                        <switch>
```

```
                            <case
condition="bpws:getVariableData(&apos;riskAssessment&apos;,&apos;risk&apos;)
!= &apos;low&apos;">
                                <empty name="Empty1">
                                    <source
linkName="Empty1_to_invokeApprover"/>
                                </empty>
                            </case>
                            <case
condition="bpws:getVariableData(&apos;riskAssessment&apos;,&apos;risk&apos;)
= &apos;low&apos;">
                                <assign name="assign">
                                    <copy>
                                    <from expression="&apos;yes&apos;"/>
                                    <to part="accept"
variable="approvalInfo"/>
                                    </copy>
                                    <source linkName="assign_to_reply"/>
                                </assign>
                            </case>
                        </switch>
                    </sequence>
                </case>
                <target linkName="receive1_to_Switch"/>
            </switch>
            <invoke inputVariable="request" name="invokeApprover"
                operation="approve" outputVariable="approvalInfo"
                partner="approver" portType="LoanApprover:LoanApproval">
                <target linkName="Empty_to_invokeApprover"/>
                <target linkName="Empty1_to_invokeApprover"/>
                <source linkName="invokeApprover_to_reply"/>
            </invoke>
        </flow>
        <receive createInstance="yes" name="receive1"
            operation="approve" partner="customer"
            portType="LoanApprover:LoanApproval" variable="request">
            <source linkName="receive1_to_Switch"/>
        </receive>
```

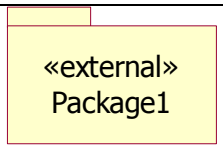
```
        <reply name="reply" operation="approve" partner="customer"
            portType="LoanApprover:LoanApproval" variable="approvalInfo">
            <target linkName="invokeApprover_to_reply"/>
            <target linkName="assign_to_reply"/>
        </reply>
    </flow>
</process>
```

Quick reference

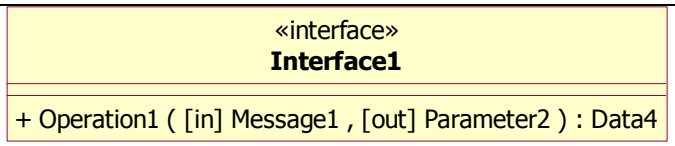
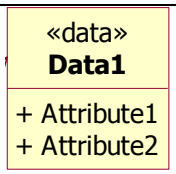
Dependency management

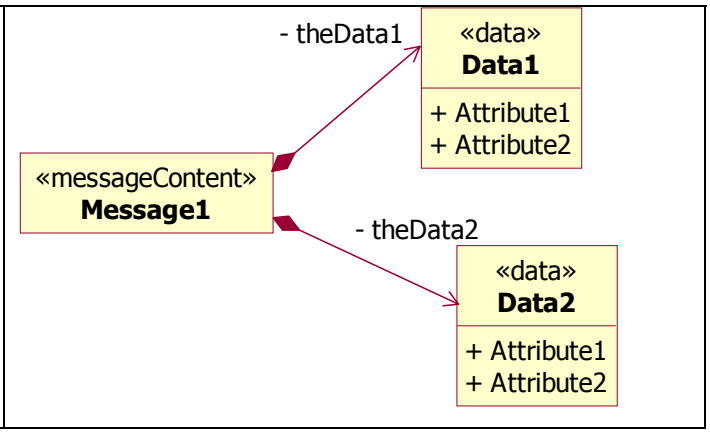
Package dependency	

External packages

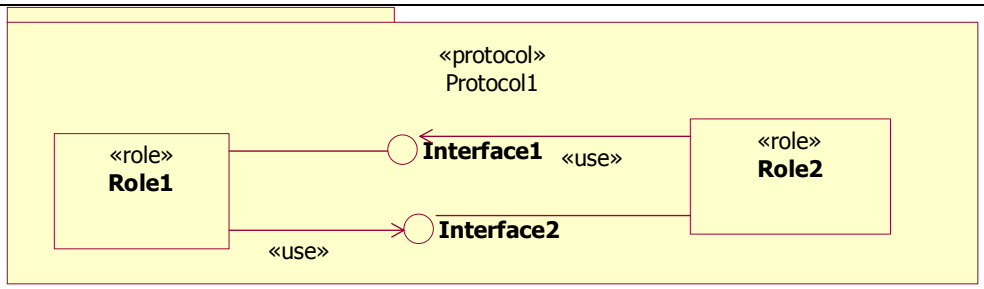
External package	

Interfaces and data types

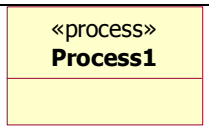
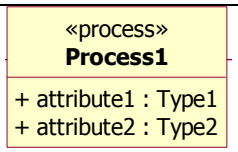
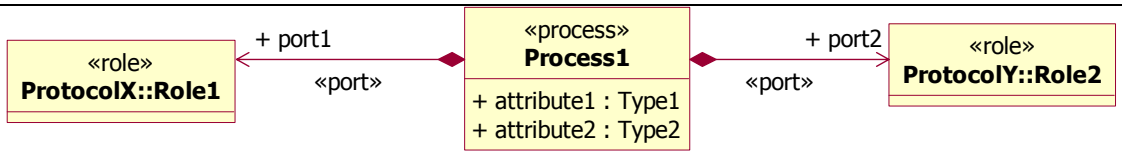
Interface	
Data types	

Message type	

Protocols

Protocol	

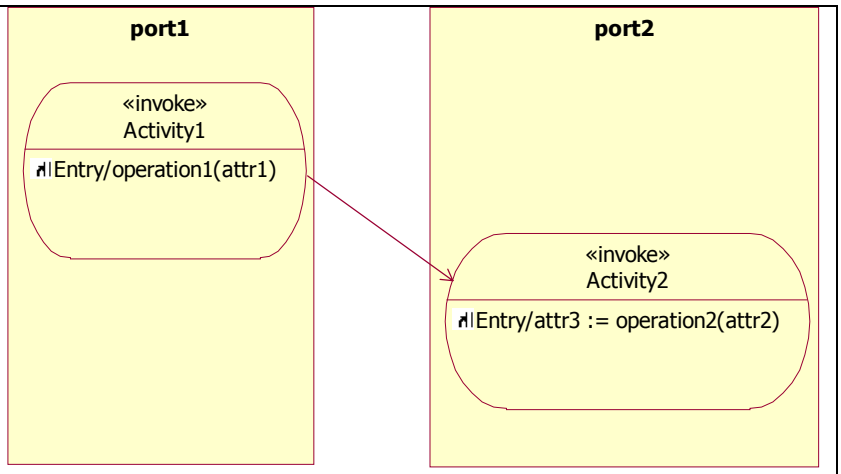
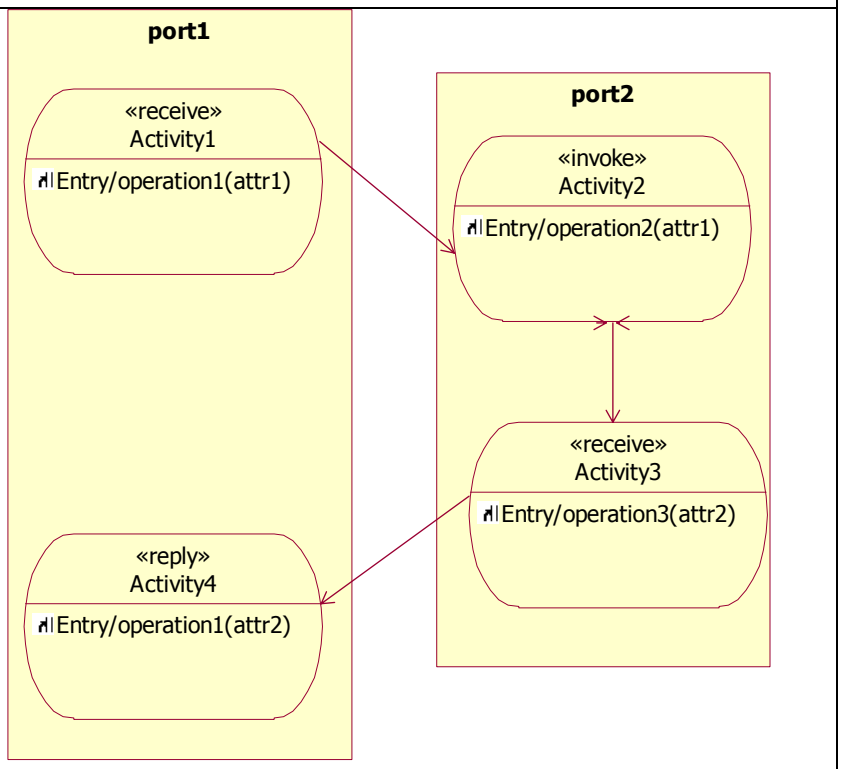
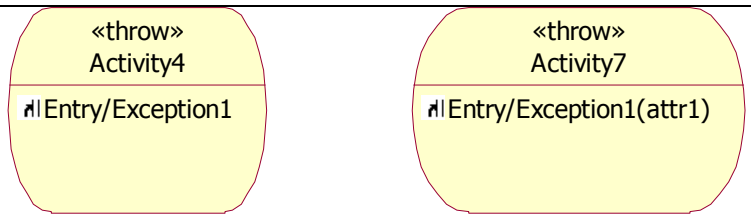
Process, state, and ports

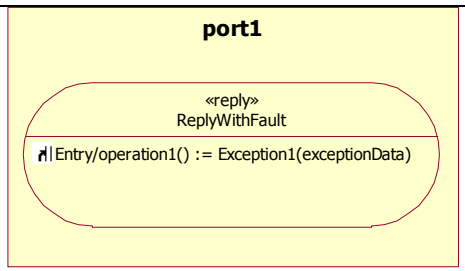
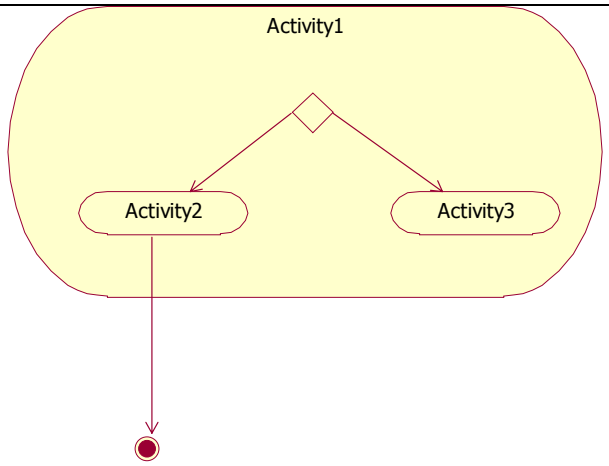
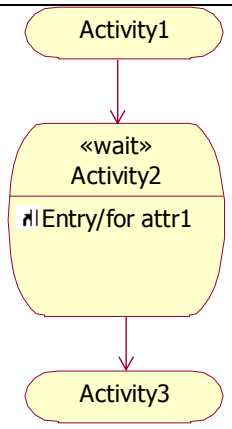
Process	
State	
Port	

Data handling

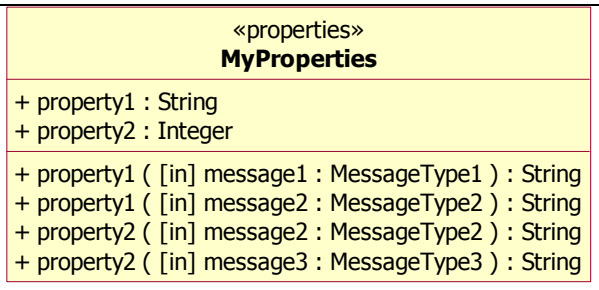
Basic activities

Comment: (SJHK) Need to provide a list of prototype expressions here.

Invoke	
Receive and reply	
Signaling exceptions	

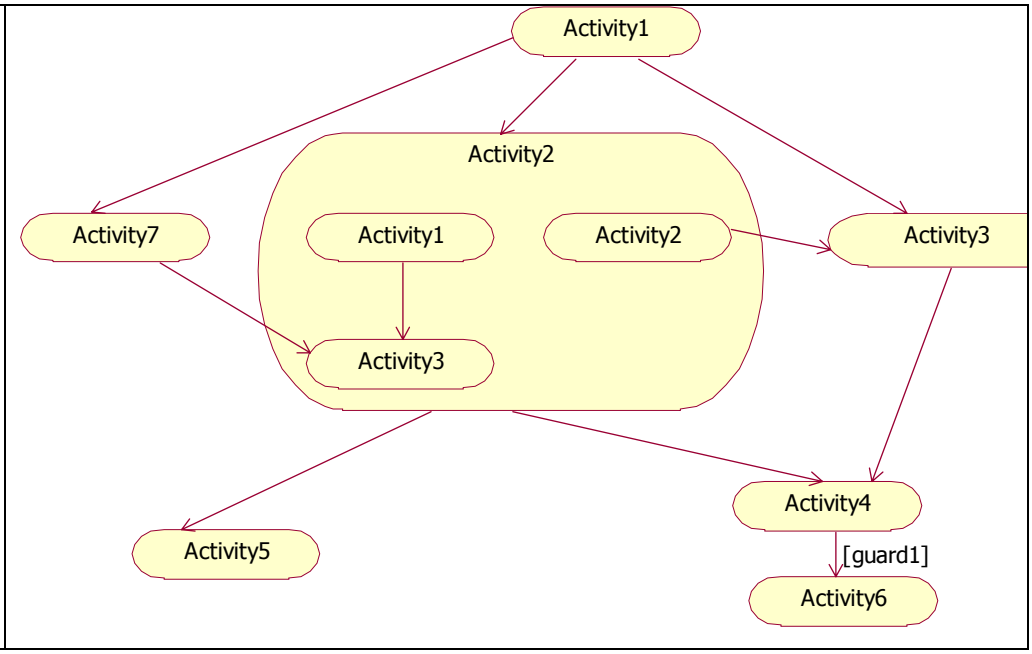
Reply with exception	
Termination	
Wait	

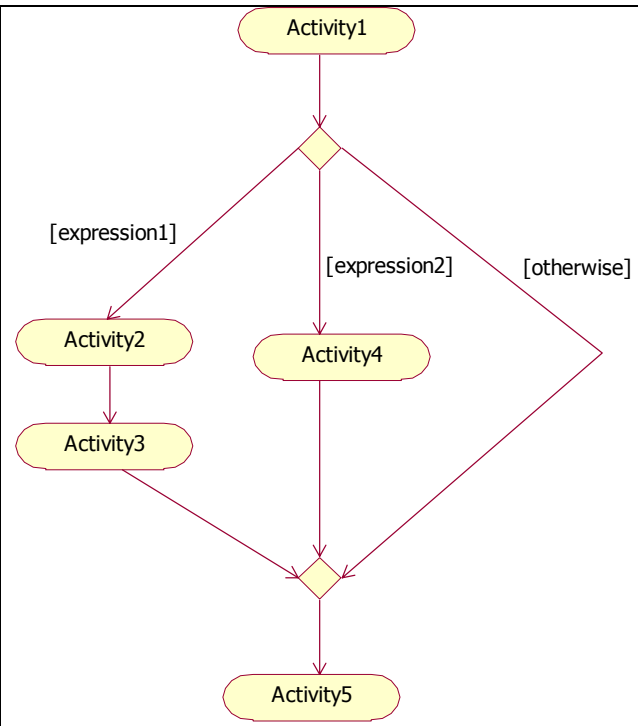
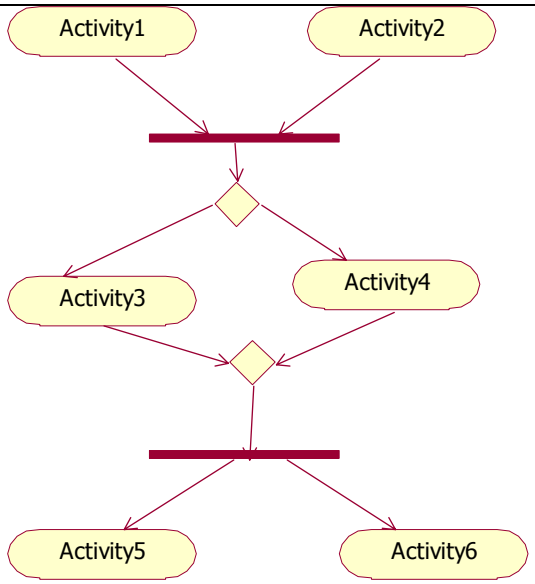
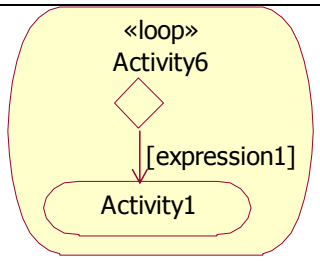
Properties and correlation

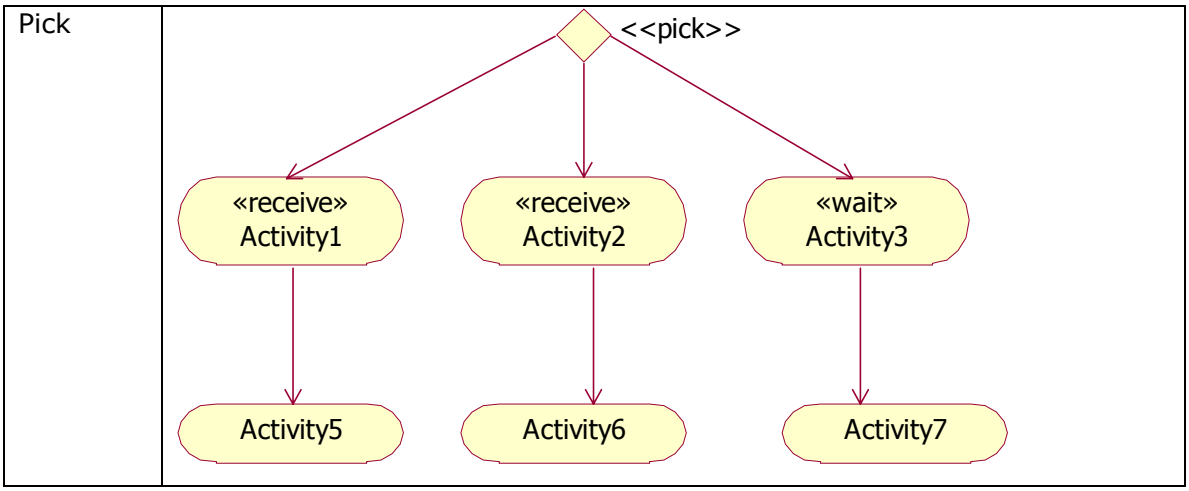
Property or property alias	

Correlation set	«correlation» **CorrelationSet1** + property1 : String + property2 : Integer
Use of correlation set by process	«process» **Process1** + «correlation» correlationSet1 : CorrelationSet1
Initializing correlation set	«receive» Activity1 Entry/correlation: initialize correlationSet1
Use of correlation sets by activity	«receive» Activity2 Entry/correlation: correlationSet1, correlationSet2
Use of correlation by invoke activity	«invoke» Activity3 Entry/out correlation: correlationSet1 Entry/in correlation: correlationSet1, initialize correlationSet2

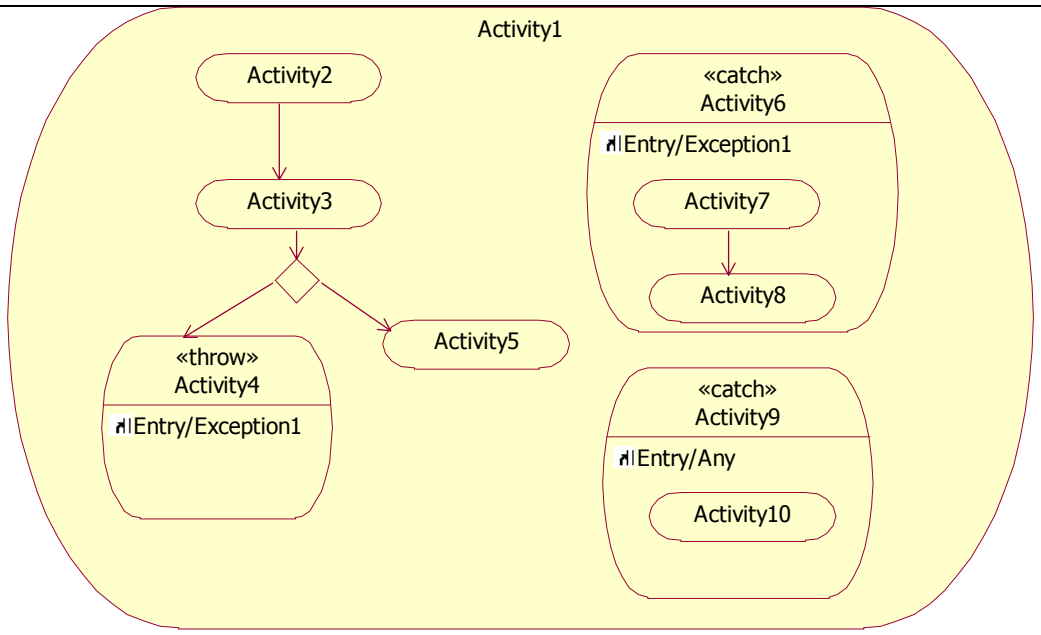
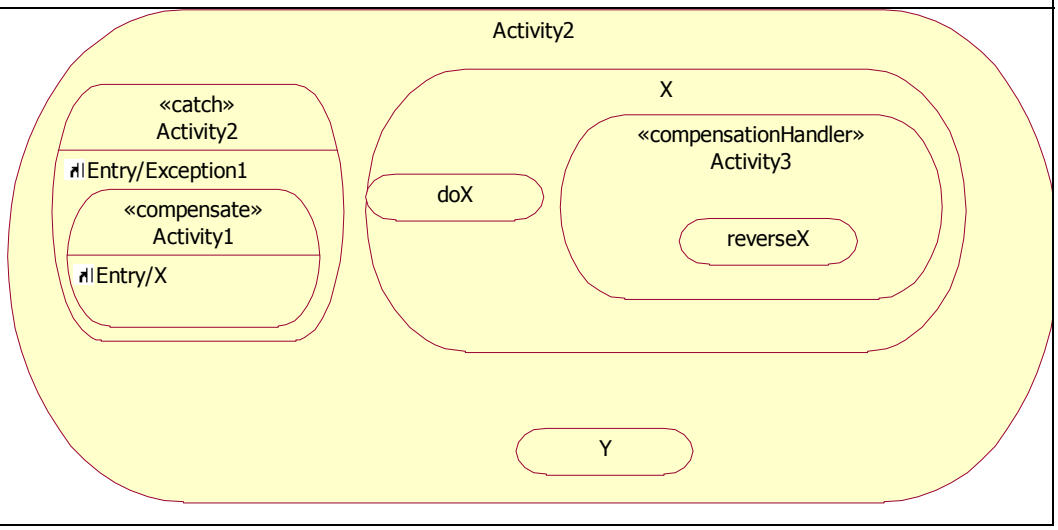
Activity coordination

Control links and nesting	

Decision and merge	
Explicit fork and join	
Loop	



Error handling

Catch and throw	
Compensation	

UML Profile Definitions

The following table outlines the UML 2.0 and BPEL 1.1 update of the profile decribed in the sections above. The profile has the same design goals as the existing profile but is able to take advantage of the modeling constructs of UML 2.0.

UML 2.0	BPEL 1.1
Class with ports	process with partner links
Activity as classifier behavior on 'process class'	activity hierarchy
Accept call action (with port specified in trigger)	receive activity (from partner)
Reply action	reply activity (to partner)
Call operation action (via port)	invoke activity (on partner)
Structured Activity Node with variables	scope with variables
Accept event action with time trigger	wait
Interruptible region with accept event actions (call and time)	pick with onMessage and onAlarm activities
Loop node	while
Conditional Node, Decision and Merge Nodes	switch
...	...

The detailed definition of this profile is under development.

Appendix B: Mapping the BPD Metamodel to BPEL4WS

This appendix presents a candidate mapping from the BPD metamodel described above to the emerging Business Process Execution Language for Web Services (BPEL4WS, or just BPEL) specification. This mapping is applicable for business analysts who want to capture their business models as platform independent models (PIM) using the BPD metamodel, and translate them to a platform specific model (PSM) for BPEL4.

The purpose of this mapping is to validate the BPD metamodel by showing how it can be mapped to a PSM for an interesting candidate platform model. Such a mapping may help improve the quality of metamodel by ensuring it can be implemented. Mappings for other platform models could also be included. The mapping is preliminary for validation purposes and requires further investigation to specify a full mapping.

BPD metamodel to BPEL4WS Mapping

The following table provides a summary of the mapping from the BPD metamodel treated as a PIM to a PSM based on BPEL as the chosen platform model (PM). Motivation for, and details of the mapping are to be supplied. It is assumed the reader is familiar with XSD, WSDL, and BPEL as Web services technologies are all part of the mapping. The mapping does not address notation or action surface language concerns.

Selecting a platform model and mapping often results in additional constraints and information being added in the form of markings to the PIM that support the chosen mapping and platform. These markings are noted in the table below, and are not considered part of the BPD metamodel.

The mapping is broken up into an number of logical segments to minimize the complexity and correspond better to Web services technologies.

Structural Mapping: XSD and WSDL

This section defines the detailed mappings between the BPD metamodel elements describing structure and the corresponding Web services model elements, XSD and WSDL.

BPD Model Element	XSD, WSDL, or BPEL4WS Construct
Package	File folder, XSD Schema file for data defined in the package, and WSDL definitions file for interfaces defined in the package.
Package nsURI (marking)	Schema target namespace, if specified. The default namespace is derived from the fully qualified package name.
Package nsPrefix (marking)	Prefix of target namespace declaration, if specified. The default prefix is the package name.

Package name	Folder name, and schema and WSDL file names, default namespace prefix name.
packageImport (<<import>> dependency between Packages)	XML namespace import in the corresponding XSD, WSDL, and BPEL documents as needed.
<<entity>> Class (isAbstract == false)	Note: <<entity>>Component mappings to XSD follow the same rules as XMI2 for a Class. XSD complexType definition, element declaration whose names are the same as the Component name.
<<entity>> Class (isAbstract == true)	XSD Complex type definition whose name is the Class name.
Enumeration	XSD Simple type that restricts string type, no corresponding WSDL portType
Property (multiplicity = 1)	Attribute declaration
Property (multiplicity > 1)	Element declaration.
Association – treated as a Property with its opposite. Information on non-navigable ends do not contribute anything to the mapping	

Process Definition: BPEL4WS Process

BPD Model Element	XSD, WSDL, or BPEL4WS Construct
<<process>> Class	A BPEL process whose name is the same as the Class name, and whose targetNamespace is derived from the fully qualified Class name. Also generates a WSDL document containing portTypes corresponding to Interfaces provided by Ports contained in the Class.
Collaboration (mapping constraint) a Collaboration can have at most two roles.	A serviceLinkType in a WSDL document. Collaboration name is the serviceLinkType name. Role names (i.e., property names) are the roles in the serviceLinkType. The type of the property specifies all the interfaces that are provided by that role. Each interface translates to a portType in the role.
ConnectableElement (a Property of the Collaboration)	A role in a serviceLinkType. The ConnectableElement name is the role name while its type specifies all the interfaces that must be provided by that role. Each interface is mapped to a portType. Whether the serviceLinkType has two roles or not depends on the number of Connections between the ConnectableElements in the Collaboration. Two-way communication results in two BPEL roles in the serviceLinkType.
CollaborationOccurrence (mapping constraint) The roleBinding must be to a Port of a	Specifies a partner element in a BPEL process. CollaborationOccurrence name is the partner name. The CollaborationOccurrence type specifies the serviceLinkType.

<<process>>Class, and the PortName and role name (i.e., Collaboration Property name) must match.	The roleBindings of the CollaborationOccurrence specify myRole and partnerRole (if needed).
Port – (mapping constraint) isBehavior must be true for ports in <<process>> Class. isService == true	The Interfaces provided by the Port are translated to WSDL portTypes as described below.
Interface provided by a Port. The type of a Port specifies all the interfaces and operations that are provided by the port.	WSDL portType for each Interface provided thought the Port. The portType contains operations of the Interface.
Operation of an Interface provided by a Port of the <<process>>Class. Note: (mapping constraint) All operation names available through a Port must be unique.	WSDL operation in the portType corresponding to the containing Interface. The portType operation name the same as the Operation name. WSDL does not support operation name overloading.
Operation. An Operation that is invoked with a CallOperationAction with isSynchronous == true must have matching AcceptCallAction and ReplyAction actions in the classifierBehavior Activity.	Corresponds to a BPEL receive activity with corresponding reply activity targeted at the same partner instance. Invoked by a request-response invoke activity that has both an input and output message.
In and inout Parameters of an Operation	WSDL input message for containing operation. Message name is derived from the Interface and operation name separated by a dash, and followed by -input. Message parts are the individual in and inout Parameters. Also creates a corresponding BPEL process variable. Variable name is same as the WSDL message name. messageType is WSDL message name.
Inout, out, and return parameters of an Operation	WSDL output message for containing operation. Message name is derived from the Interface and operation name separated by a dash, and followed by -output. Message parts are the individual out, return and inout Parameters. BPEL process variable as for in parameters.
Operation raised Exceptions	WSDL message and fault message for the operation. Message name is the class name followed by the operation name separated by a dash and followed by –fault. Message parts are the attributes of the Exceptions raised. Each part name is the exception name converted to lower case.

	BPEL process variable whose name is the same as the WSDL fault message name. Can be used for faultVariable in catch element of a faultHandler

Behavioral Mapping: BPEL4WS

BPD Model Element	XSD, WSDL, or BPEL4WS Construct
Activity as the <<process>> Class::classifierBehavior	BPEL process
<<compensation>> StructuredActivityNode in an Activity.	compensate activity
Object and process identity - self and <<id>> Properties	XSD keys and BPEL correlationSets. Each operation of a portType has as the first parts of its input message values corresponding to the id properties of the containing Class. In the BPEL process, a correlationSet with the appropriate property and propertyAlias declarations is used to define the necessary correlation.
Properties in a <<process>>Class	BPEL variable in the process whose name is the Property name, and whose type is it's a WSDL message containing a part whose type is the XSD element corresponding to the Property's type.
StructuredActivityNode	Scope
Variable - Local variables in a StructuredActivityNode	variables in the scope. Variables that are represented as a Variable in a StructuredActivityNode have their WSDL message types defined inline with the variable declaration.
CallOperationAction in an ActivityPartition. (mapping constraint) The ActivityPartition must refer to a Port of the <<process>>Class.	BPEL invoke activity of the corresponding WSDL operation through the partner corresponding to the Port.
ForkNode and JoinNode pairs.	BPEL flow activities
ActivityEdge	BPEL sequence and links for concurrency control
AcceptCallAction in an ActivityPartition. (mapping constraint) The ActivityPartition must refer to a Port of the <<process>>Class.	BPEL receive activity of the corresponding WSDL operation through the partner corresponding to the Port.
ReplyAction (mapping constraint) The ReplyAction must be in the same ActivityPartition as it's corresponding CallOperationAction.	BPEL reply activity
AcceptEventAction	BPEL event handler
Read property, link, and/or variable actions	BPEL getVariableData or getVariableProperty functions.
Write property, link, and variable actions	BPEL assign activity
RaiseExceptionAction	BPEL throw activity and WSDL fault

	message
An ActivityEdge leading to an ActivityFinalNode	terminate activity
AcceptEventAction with TimeTrigger	wait activity
MergeNode	empty activity with links as needed, or join??
Multiple, guarded ControlFlows out of an Action, DecisionNode, or ConditionalNode	switch activity
Guarded ControlFlow	Flow link transition condition
LoopNode.	while activity
AcceptCallActions and CallTrigger on a message or AcceptEventActions and a TimeTrigger in a <<pick>InterruptibleActivityRegion.	BPEL pick activity
ExceptionHandler	BPEL fault handler

Appendix C: Mapping EDOC to BPD Profile

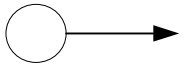
Comment: we hope that with the inclusion of the Borland, Data Access, EDS and 88 Solutions team we can have a mapping developer for inclusion in this submission.

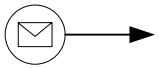
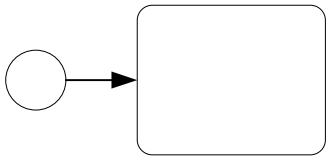
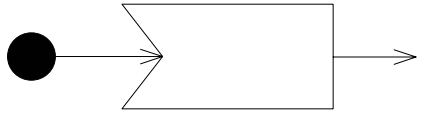
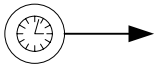
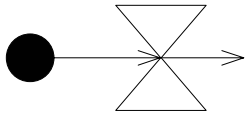
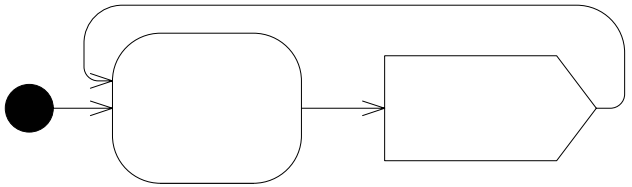
Appendix D: Mapping BPMN to BPD Profile

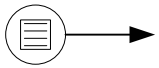
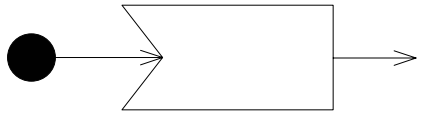
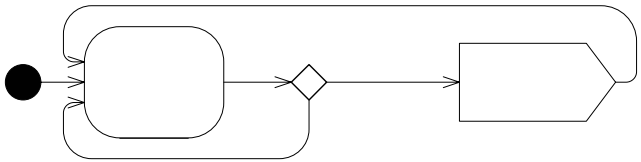
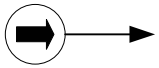
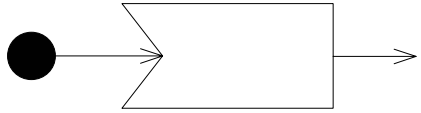
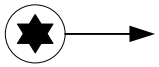
Members of bpmi.org and the OMG are interested in the unification of the UML 2.0 and BPMN notation for the support of the business user. This draft mapping is in support of this dialog and will be completed in a revised submission. Note, it may be the case that provision of a new, business-level notation will require changes to the UML 2.0 metamodel; these will be documented in any case where they arise.

Mapping Overview

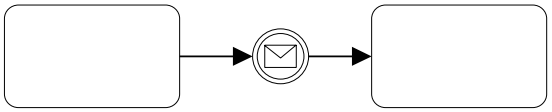
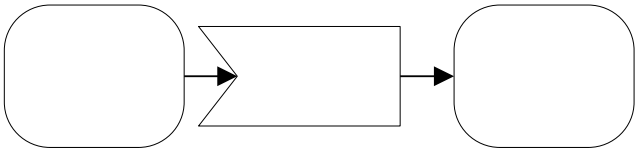
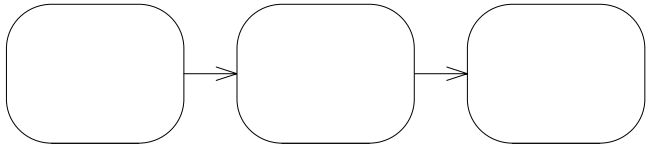
The following table outlines the mapping defined from the BPMN notation to the BPD meta model in detail.

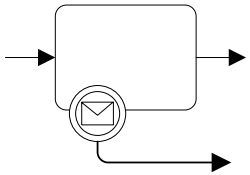
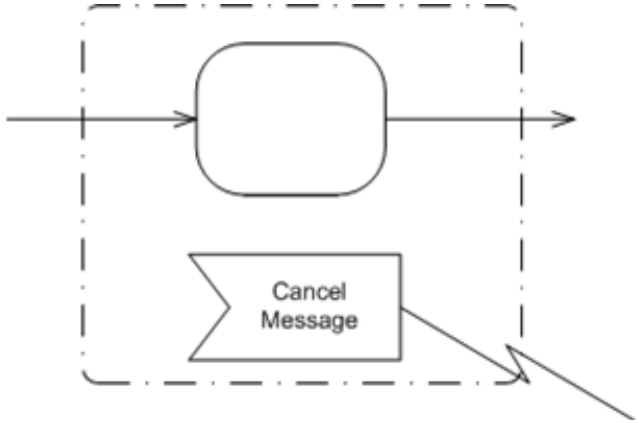
Element/Concept	Description	BPMN Notation	Activity Diagram Notation
Event	An event is something that "happens" during the course of a business process. These events affect the flow of the process and usually have a cause (trigger) or an impact (result). There are three types of Events, based on when they affect the flow: Start, Intermediate, and End.		Start Node, End Node, Signal, Connector, and various design patterns. See the rows that follow that deal with Events.
Start (None, Message, Timer, Rule, Link, Multiple)	As the name implies, the Start Event indicates where a particular process will start. Start Events can have "Triggers" that define the cause for the event. There are multiple ways that these events can be triggered.		The Start Event maps to the Start Node plus some design patterns. See next five rows…  It should be noted that if a receipt of a signal is used in the mapping, then the start node, followed by a control flow is not required.

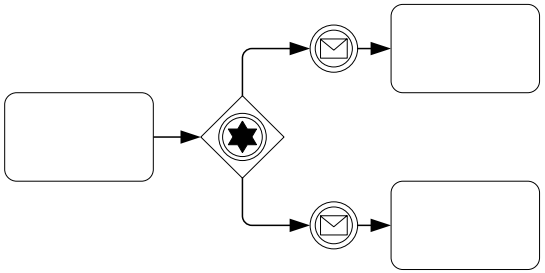
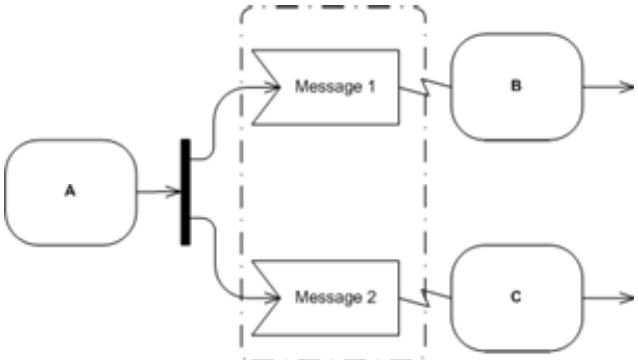
Message Trigger	The message is generally sent from an outside source and triggers the start of the process. The receipt of the message needs to be configured the same way as a receive type of activity.	 In BPMN, this is also equivalent to a None Start Event followed by a Receive Task. 	Start Node followed by a Signal Receipt AcceptEventAction Initial node is optional (If no incoming transitions, the action starts when the enclosing activity starts). 
Timer Trigger	The process is started whenever a timer reaches a specific time/date or a recurring time/date.		The Timer Start Event is mapped to Start Node followed by a AcceptEventAction where the trigger is time event.  To support this signal, a separate Process must be created that runs infinitely and has a timer to send the Signal. A no action activity implemented as a timer to go off on a specific time and date or on a cycle. 

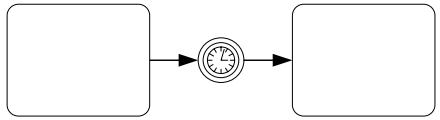
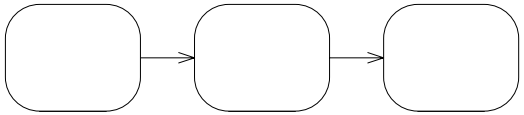
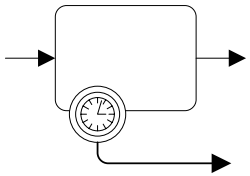
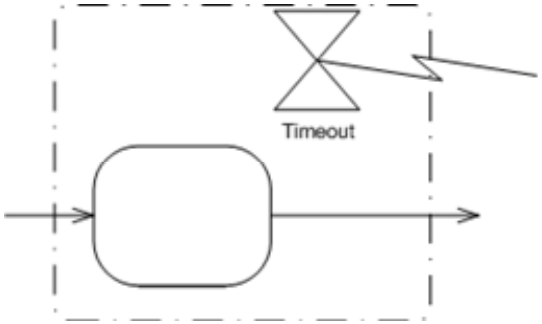
Rule Trigger	This indicates that a Signal is sent from another Process, based on the satisfaction of a Rule, should start the current process. There will be a corresponding End Event in another Process.		The Rule Start Event is mapped to Start Node followed by a receive Signal.  To support this signal, a separate Process must be created that runs infinitely and has an activity that checks data and a decision as to whether or not to send a signal. 
Link Trigger	This indicates that a Signal is sent from another Process should start the current process. There will be a corresponding End Event in another Process.	This indicates that a Signal is sent from another Process to start the current process. There will be a corresponding End Event in another Process. 	The Link Start Event is mapped to Start Node followed by a receive Signal. 
Multiple Trigger	Any number or combination of the above types of Start Events. Any one of the specified Triggers will start the Process.		Combinations of above, depending Event settings

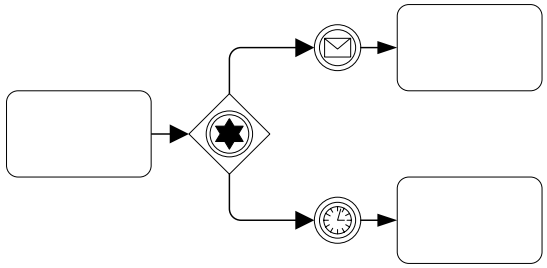
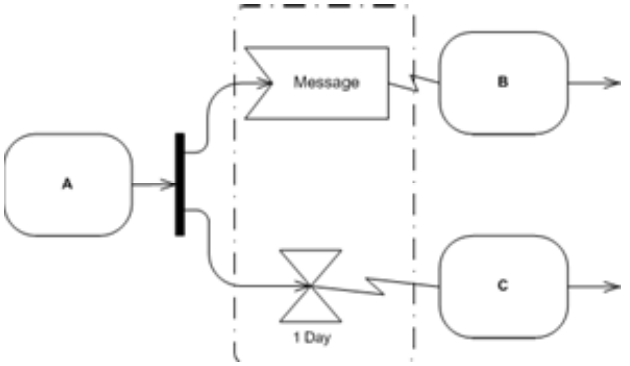
Intermediate (None, Message, Timer, Exception, Cancel, Compensation, Rule, Link, Multiple, Branching)	Intermediate Events occur between a Start Event and an End Event. It will affect the flow of the process, but will not start or (directly) terminate the process. Intermediate Events have "Triggers" that define the cause for the event. There are multiple ways that these events can be triggered.		Signals, Connectors, and various design patterns. See next 26 rows…
Message Trigger	The trigger is the receipt of a message from an outside source (outside of the Process).		See next three rows…

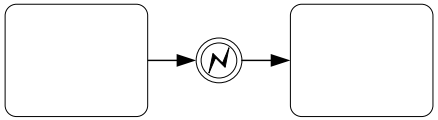
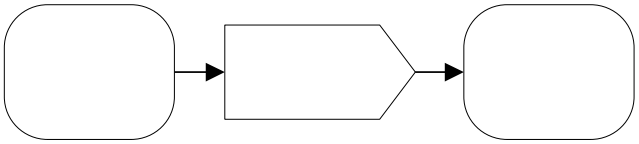
Within Normal Flow	When used in this form, the Message Intermediate Event acts as a place where a message is expected from an outside source. The Process (or that particular path) will wait for the message to arrive before it continues.	 In BPMN, this is also equivalent to a Receive Task. 	The Message Intermediate Event in flow can map to a Receive Signal.  Alternatively, an activity that receives the message can be used. 

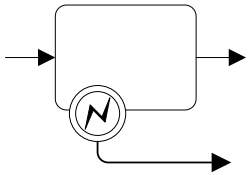
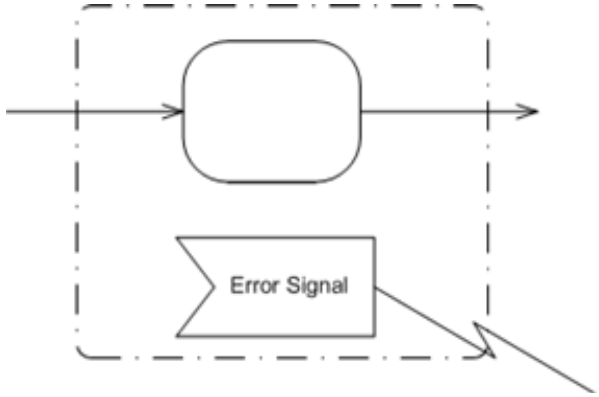
Attached to Activity Boundary	When used in this form, the Message Intermediate Event acts as a mechanism to interrupt an activity. If the activity is running and the particular message is received, then the activity will be stopped and the flow will be diverted to the Sequence Flow that exits the Intermediate Event. The configuration of the receipt of the message is basically the same as configuring an activity that receives a message.	The Intermediate Event is physically attached to the boundary of the activity. 	To create this behavior, an interruptible region is created to surround the activity. Also, a Receive Signal is placed within the region. An interrupting edge is used to exit the region (and the activity) with the signal arrives. 

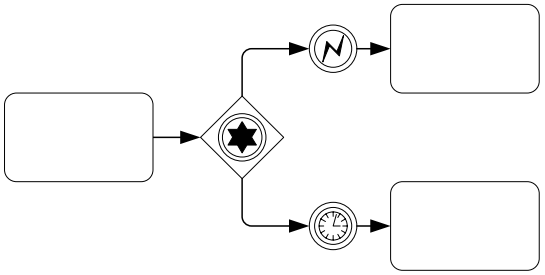
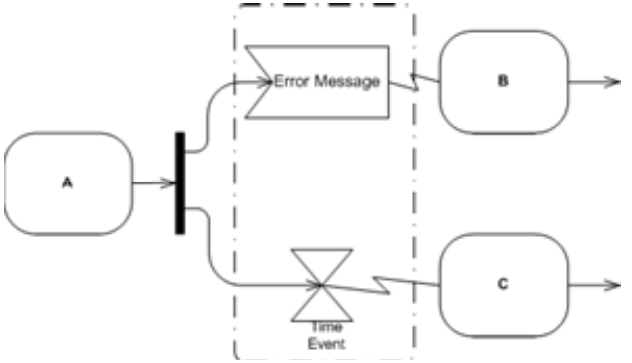
Within Event-Based Decision	When used in this form the Message Intermediate Event is a part of decision pattern that is based on Events that occur during the process. In this case, the first message that arrives will determine that path that is chosen. The occurrence of one message will exclude the other path(s).		The Message Intermediate Events will map to the receipt of separate Signals. All the signals to be received by the pattern must be available at the same time, so they follow a fork node. But only one will be used—to the exclusion of all others. To do this, the signal receipts are all placed within an interruptible region. When a signal is received, this will lead to an interrupting edge that will cause the closing of the region, thus excluding the other signal receipts. 
Timer Trigger	This event is triggered when the process time reaches the time specified in the attributes of the Event. The time may be specified as a recurring time/date or a relative time/date.		See next three rows…

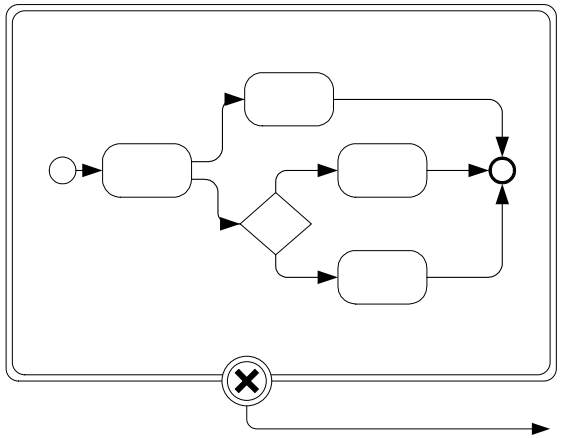
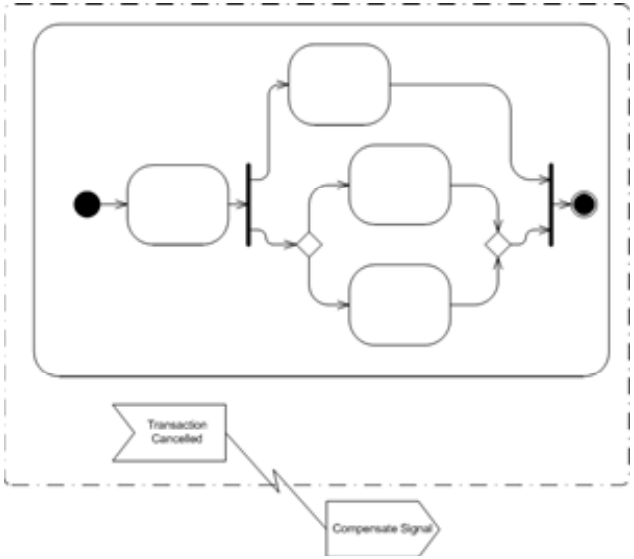
Within Normal Flow	When used in this form, the Timer Intermediate Event acts as a place a delay is expected. The Process (or that particular path) will wait until the process time reaches the time specified in the attributes of the Event.		The Timer Intermediate Event is mapped to an activity that is implemented as a Timer. [Add a delay stereotype to metamodel.] 
Attached to Activity Boundary	When used in this form, the Timer Intermediate Event acts as a mechanism to interrupt an activity. If the activity is running and the specified time is reached, then the activity will be stopped and the flow will be diverted to the Sequence Flow that exits the Intermediate Event.	The Intermediate Event is physically attached to the boundary of the activity. 	To create this behavior, an interruptible region is created to surround the activity. Also, an AcceptEventAction where the trigger is time event is placed within the region. An interrupting edge is used to exit the region (and the activity) with the signal arrives. 

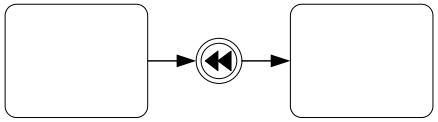
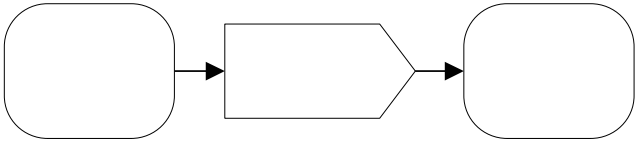
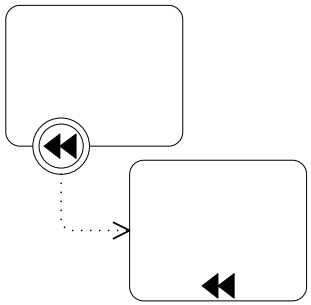
Within Event-Based Decision	When used in this form the Timer Intermediate Event is a part of decision pattern that is based on Events that occur during the process. In this case, the first event, either the Message or the time event, that arrives will determine that path that is chosen. The occurrence of one event will exclude the other path(s).		The Timer Intermediate Event will map to the receipt of a Signal that is an AcceptEventAction where the trigger is time event. All the signals to be received by the pattern must be available at the same time, so they follow a fork node. But only one will be used—to the exclusion of all others. To do this, the signal receipts are all placed within an interruptible region. When a signal is received, this will lead to a interrupting edge that will cause the closing of the region, thus excluding the other signal receipts. 
Exception Trigger	This Event either reacts to or generates an exception, depending on the use of the Event in the Process.		See next three rows…

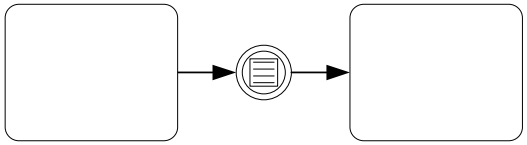
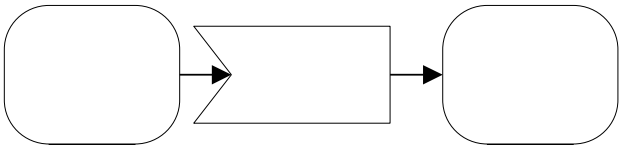
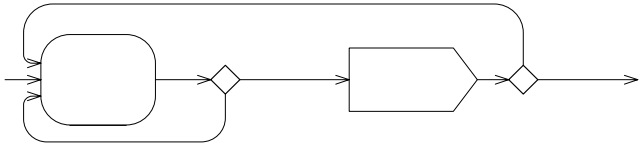
Within Normal Flow	When used in this form, the Exception Intermediate Event acts as a place a where an exception should be triggered (e.g., the exception is thrown). The Process will trigger the exception and then continue to the next activity.		The Exception Intermediate Event is mapped to a Send Signal of type Exception. RaiseExceptionAction <Review all EventActions.> 

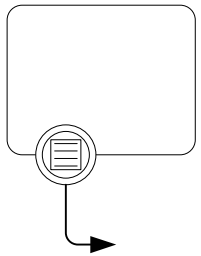
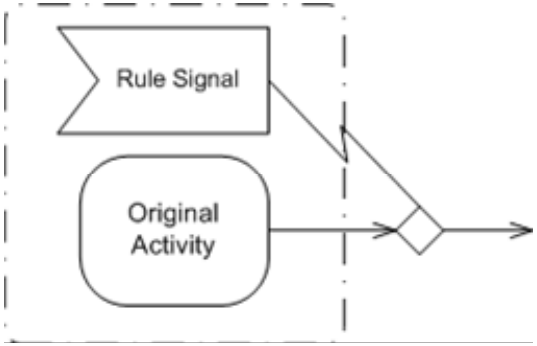
Attached to Activity Boundary	When used in this form, the Exception Intermediate Event acts as a mechanism to interrupt an activity. If the activity is running and the specified exception has been triggered, either through an application message or through another Intermediate Event that "throws" the exception, then the activity will be stopped and the flow will be diverted to the Sequence Flow that exits the Intermediate Event.	The Intermediate Event is physically attached to the boundary of the activity. 	To create this behavior, an interruptible region is created to surround the activity. Within the region, a signal receipt is added to catch the signal that was created elsewhere. An interrupting edge that exits the signal is used to exit the region (and the activity) when the signal arrives.  Note: UML also provides an exception handling mechanism that creates protected nodes and specific activities to be used when an exception occurs. This mechanism is more specific than the patterns created by a BPMN model. BPMN uses a more general capability of diverting flow when an exception occurs.

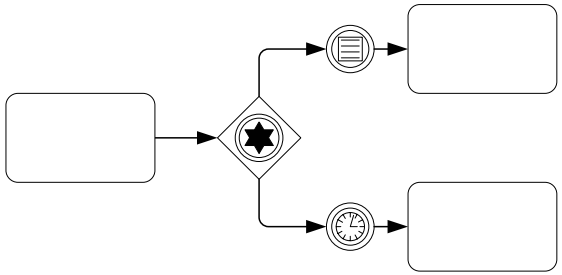
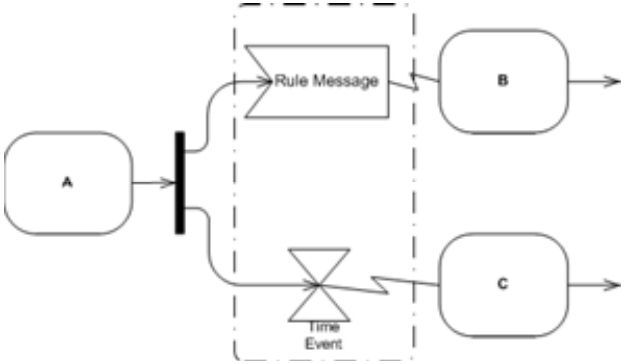
Within Event-Based Decision	When used in this form the Exception Intermediate Event is a part of decision pattern that is based on Events that occur during the process. In this case, the first event, either the exception or the time event, that arrives will determine that path that is chosen. The occurrence of one event will exclude the other path(s).		The Exception Intermediate Event will map to the receipt of a Signal. All the signals to be received by the pattern must be available at the same time, so they follow a fork node. But only one will be used—to the exclusion of all others. To do this, the signal receipts are all placed within an interruptible region. When a signal is received, this will lead to a interrupting edge that will cause the closing of the region, thus excluding the other signal receipts. 
Cancel Trigger	This Event only applies to Sub-Processes that are defined as Transactions.		See next row…

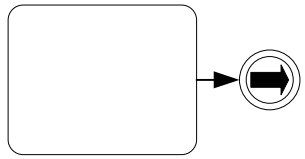
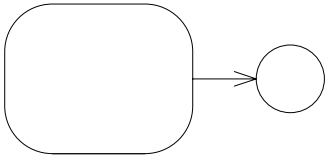
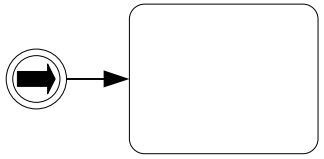
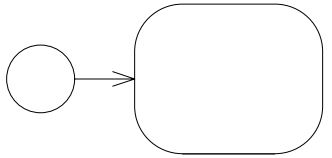
Attached to Activity Boundary	This is the only form that this Event can be used. When attached to the boundary of a Transaction Sub-Process, the Cancel Intermediate Event acts as a mechanism to interrupt an activity. If the activity is running and the specified cancel notification has been triggered, either through an application message, transaction protocol, or through a Cancel End Event within the Transaction, then the activity will be stopped and the flow will be diverted to the Sequence Flow that exits the Intermediate Event.		To use the Cancel Intermediate Event, an interruptible region encloses a Sub-Process. A receive signal is used to interrupt the region. If some mechanism for roll-back, including compensation is required, then a Compensate Signal should be sent after the Transaction has been interrupted. 
Compensation Trigger	This Event either reacts to or generates a compensation, depending on the use of the Event in the Process.		See next two rows…

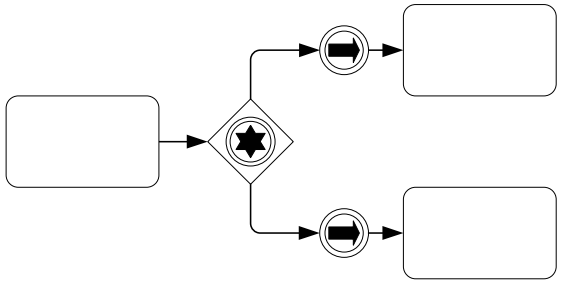
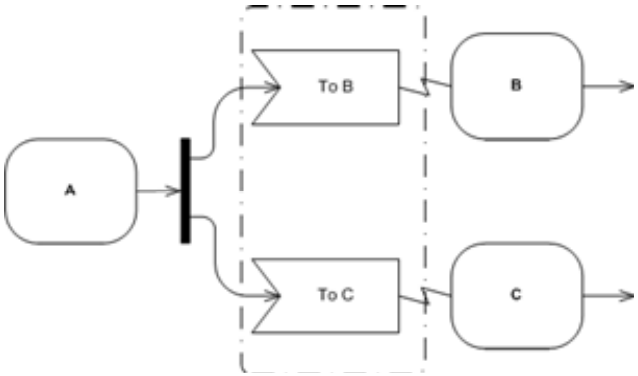
Within Normal Flow	When used in this form, the Compensation Intermediate Event acts as a place a where a compensation should be triggered (e.g., the compensation is thrown). The Process will trigger the compensation and then continue to the next activity.		This could be a signal. The receipt of the signal should be in a separate process that is ready to receive it. 
Attached to Activity Boundary	When used in this form, the Compensation Intermediate Event acts as a mechanism to identify the activity that will be used to compensate for the original performance of the activity. If the activity *has been completed* and the specified compensation has been triggered, then the (compensation) activity that is Associated with the original activity will be performed.		See Compensation Association.

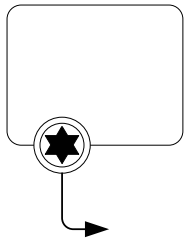
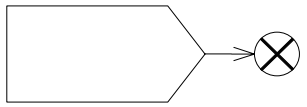
Rule Trigger	This event is triggered when a business rule becomes true during the performance of the process.		See next three rows…
Within Normal Flow	When used in this form, the Rule Intermediate Event acts as a place where a delay will occur until a specific rule has been satisfied. The Process (or that particular path) will wait for the rule to be true before it continues.		This map to a signal that arrives when the rule is determined to be true. Could be guard – a guard will wait. <Check>  To support this signal, a separate Process must be created that runs in parallel and has an activity that checks data and a decision as to whether or not to send a signal. 

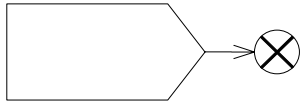
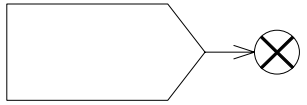
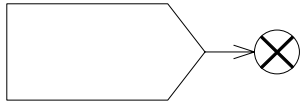
Attached to Activity Boundary	When used in this form, the Rule Intermediate Event acts as a mechanism to interrupt an activity. If the activity is running and the particular rule is satisfied, then the activity will be stopped and the flow will be diverted to the Sequence Flow that exits the Intermediate Event.		To create this behavior, an interruptible region is created to surround the activity. Also, a Receive Signal is placed within the region. An Interrupting edge is used to exit the region (and the activity) with the signal arrives. The Interrupting Edge and the normal edge that exits the activity are merged since only one of them will happen.  To support this signal, a separate Process must be created that runs in parallel and has an activity that checks data and a decision as to whether or not to send a signal (see row above).

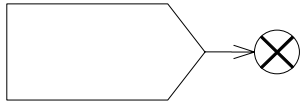
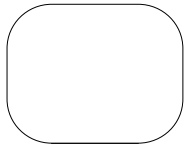
Within Event-Based Decision	When used in this form the Rule Intermediate Event is a part of decision pattern that is based on Events that occur during the process. In this case, the first event, either the rule being satisfied or the time event, that arrives will determine that path that is chosen. The occurrence of one event will exclude the other path(s).		The Rule Intermediate Event will map to the receipt of a Signal. All the signals to be received by the pattern must be available at the same time, so they follow a fork node. But only one will be used—to the exclusion of all others. To do this, the signal receipts are all placed within an interruptible region. When a signal is received, this will lead to a interrupting edge that will cause the closing of the region, thus excluding the other signal receipts. 
Link Trigger	This Event either acts as a "Go To" object or receives a signal generated by another Process, depending on the use of the Event in the Process.		See next two rows…

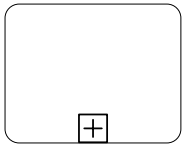
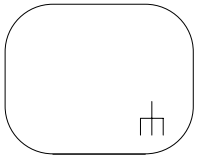
Within Normal Flow			This maps to an Activity Edge Connector. 
Within Normal Flow			This maps to an Activity Edge Connector. 

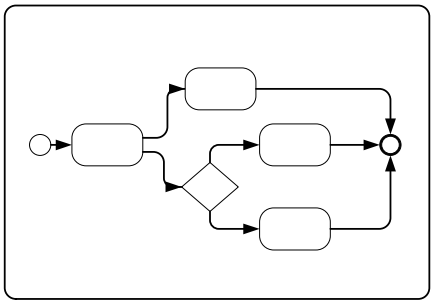
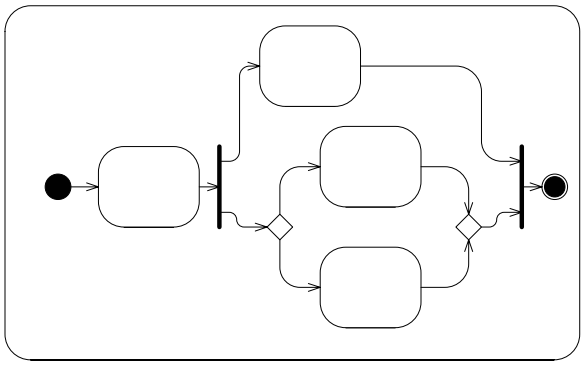
Within Event-Based Decision	When used in this form the Link Intermediate Event is a part of decision pattern that is based on Events that occur during the process. In this case, the first event that arrives will determine that path that is chosen. The occurrence of one Event will exclude the other path(s).		The Link Intermediate Event will map to the receipt of a Signal. All the signals to be received by the pattern must be available at the same time, so they follow a fork node. But only one will be used—to the exclusion of all others. To do this, the signal receipts are all placed within an interruptible region. When a signal is received, this will lead to a interrupting edge that will cause the closing of the region, thus excluding the other signal receipts. 
Multiple Trigger	This Event can only be used as a mechanism to interrupt activities.		See next row…

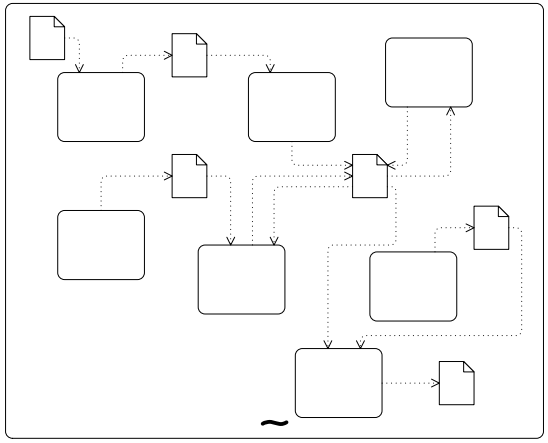
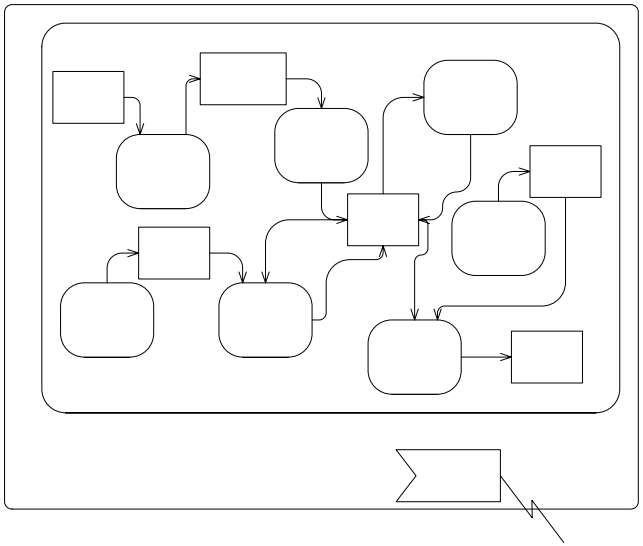
Attached to Activity Boundary	This is the only form that this Intermediate Event can be used. When attached to the boundary of an activity, the Multiple Intermediate Event acts as a mechanism to interrupt an activity. If the activity is running and *any one* of the specified triggers occurs, then the activity will be stopped and the flow will be diverted to the Sequence Flow that exits the Intermediate Event.		Combinations of above, depending Event settings
End (None, Message, Exception, Cancel, Compensation, Link, Terminate, Multiple)	As the name implies, the End Event indicates where a process will end. End Events may define a "Result" that is a consequence of a Sequence Flow ending. These Results may require some processing before the Process can be completed.		Activity Final, Flow Final. See next seven rows… 
Message Result	A message is sent before the Process (or path) is completed.		This maps to a Signal that is followed by a flow final node. 

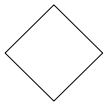
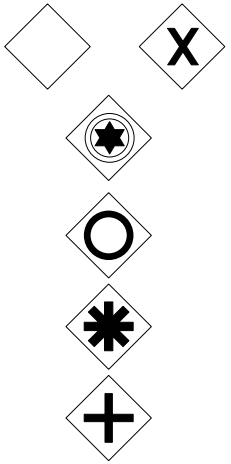
Exception Result	An exception is "thrown" before the Process (or path) is completed.		This maps to a signal that is followed by a flow final node. It will be paired with a signal receipt that is with an interruptible region. See Exception Trigger. 
Cancel Result	This is only used within a Sub-Process that is set as a Transaction. A cancel notification is "thrown" and the transaction will be stopped and rolled back.		This maps to a signal that cancels a transaction that is followed by a flow final node. It will be paired with a signal receipt that is contained an interruptible region. See Cancel Trigger. 
Compensate Result	A Compensation notice is "thrown" before the Process (or path) is completed.		This maps to a compensation signal that is followed by a flow final node. It will be paired with a signal receipt for this signal. This receipt will likely start another activity used for compensation. See Compensation Association. 

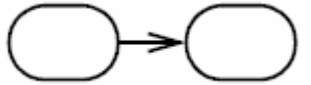
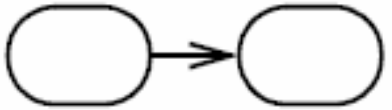
Link Result	This indicates that a Signal will be sent to another Process. There will be a corresponding Start Event in another Process.		This maps to a Signal that is followed by a flow final node. This will be paired with a corresponding receipt signal that will start another activity. See Link Trigger. 
Multiple Result	All of the above Results (except Cancel) may be used in combination and any quantity. All specified results will be processed before the Process (or path) is completed.		Combinations of the above End Events, depending Event settings
Terminate Result	This End Event will signal the end of the Path and will cause the termination (without compensation) of any other path that may still be active.		This maps to the Activity Final 
Task (Atomic)	A Task is an atomic activity that is included within a Process. A Task is used when the work in the Process is not broken down to a finer level of Process Model detail.		This maps to an Action for all TaskTypes except Referenced or an Activity with a CallBehaviorAction for the TaskType Referenced. 

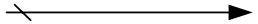
Process/Sub-Process (non-atomic)	A Sub-Process is a compound activity that is included within a Process. It is compound in that it can be broken down into a finer level of detail (a Process) through a set of sub-activities.	See Next Two Figures	See next to rows…
Collapsed Sub-Process	The details of the Sub-Process are not visible in the Diagram. A "plus" sign in the lower-center of the shape indicates that the activity is a Sub-Process and has a lower-level of detail.		This maps to an Activity with a CallBehaviorAction (for referenced—SubProcessType is Independent) or StructuredActivityNode (for nested—SubProcessType is Embedded). 

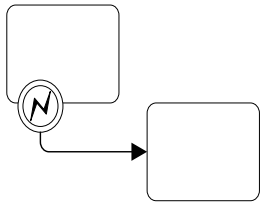
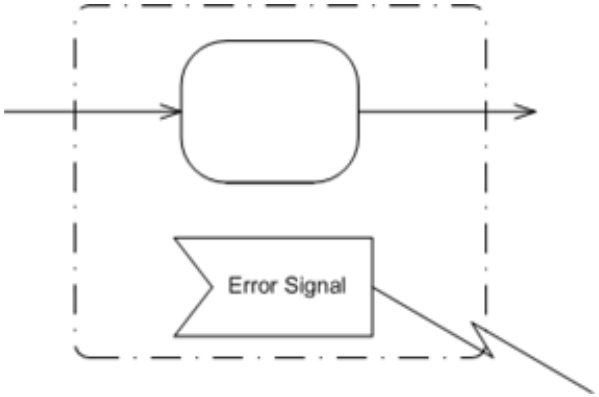
Expanded Sub-Process	The boundary of the Sub-Process is expanded and the details (a Process) are visible within its boundary. Note that Sequence Flow cannot cross the boundary of a Sub-Process.		This maps to an Activity with a CallBehaviorAction (for referenced—SubProcessType is Independent) or StructuredActivityNode (for nested—SubProcessType is Embedded). 

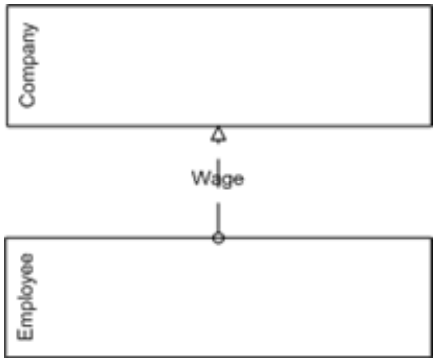
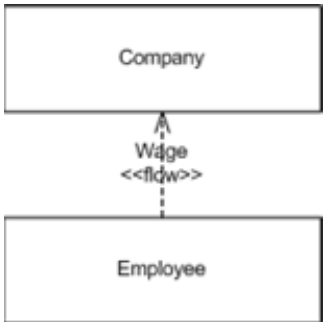
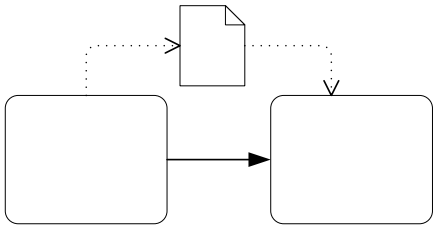
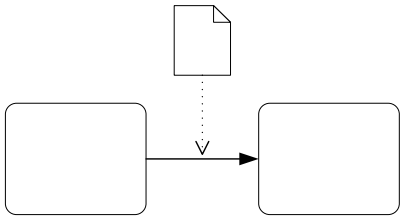
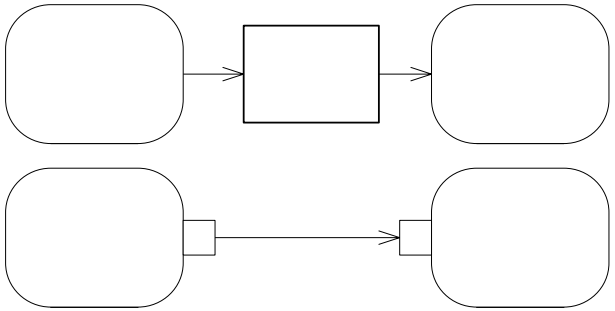
Ad-Hoc Process	This type of Process is a collection of activities that has no specific sequence or multiplicity. Presumably, the performers of the activities will determine when and how often the activities will occur. In addition, there is some condition that will determine that the Process as completed.	The activities in this process can be done in almost any order. But there are some dependencies as shown by the input and output data objects. But the “data flow” does not imply the sequence of performance. For example, The “Write Text” Task produces a text draft, but the next activity may be the “Generate Graphics” or “Organize References” Task. 	This maps to an Activity that has Sub-Activities. The sub-activities are not organized in with any control flow. However, data flow may be used. [Unfortunately (for Ad-Hoc) Data flow also contains control flow, which creates unwanted implications of control]  This particular type of activity does not map to an execution level, such as BPEL. This needs to be supported by an activity somewhere that will monitor the condition of the Ad-Hoc Process and then send a signal that will be used to interrupt the interruptible region.

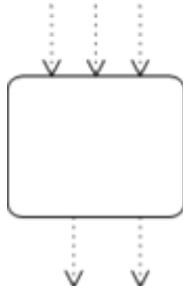
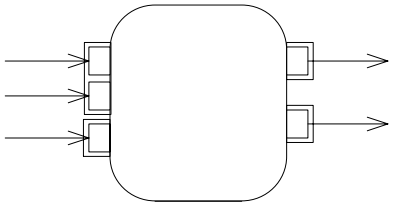
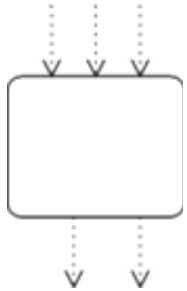
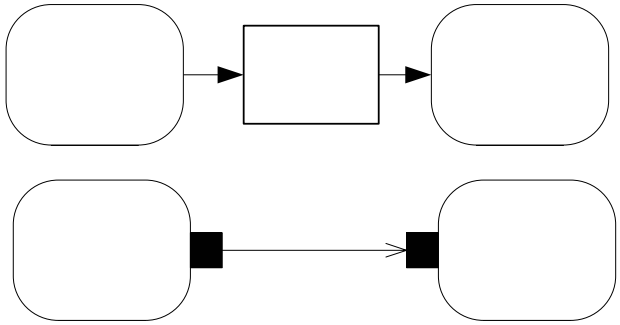
Gateway	A Gateway is used to control the divergence and convergence of multiple Sequence Flow. Thus, it will determine branching, forking, merging, and joining of paths.		
Gateway Control Types	Icons within the diamond shape will indicate the type of flow control behavior. The types of control include: • XOR -- exclusive decision and merging. Both Data-Based and Event-Based. Data-Based can be shown with or without the "X" marker. • OR -- inclusive decision • Complex -- complex conditions and situations (e.g., 3 out of 5) • AND -- forking and joining Each type of control affects both the incoming and outgoing Flow.		See the sections on Decisions, Merges, Forks, and Joins
Sequence Flow	A Sequence Flow is used to show the order that activities will be performed in a Process.	See next seven figures	

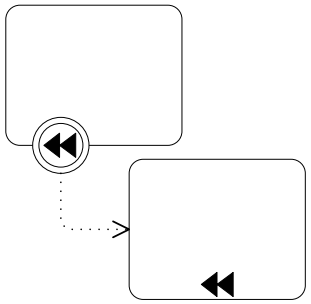
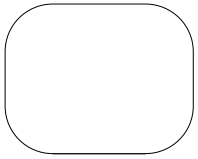
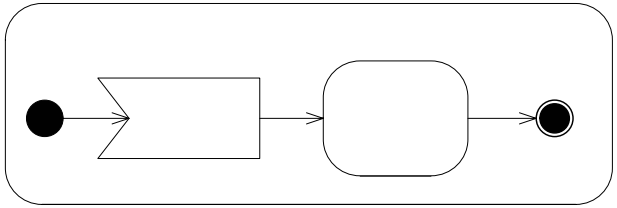
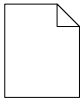
Normal flow	Normal Sequence Flow refers to the flow that originates from a Start Event and continues through activities via alternative and parallel paths until it ends at an End Event.		Control Flow 
Uncontrolled flow	Uncontrolled flow refers to flow that is not affected by any conditions or does not pass through a Gateway. The simplest example of this is a single Sequence Flow connecting two activities. This can also apply to multiple Sequence Flow that converge on or diverge from an activity. For each uncontrolled Sequence Flow a "Token" will flow from the source object to the target object.		Control Flow 

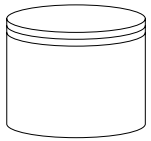
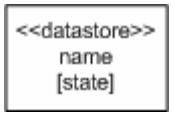
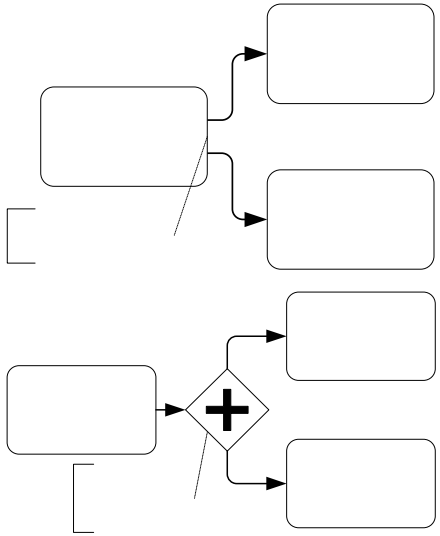
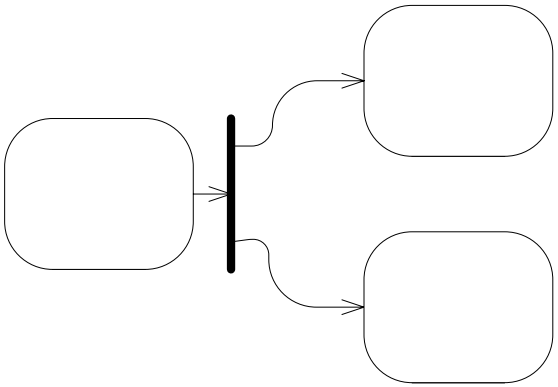
Conditional flow	Sequence Flow can have condition expressions that are evaluated at runtime to determine whether or not the flow will be used. If the conditional flow is outgoing from an activity, then the Sequence Flow will have a mini-diamond at the beginning of the line (see figure to the right). If the conditional flow is outgoing from a Gateway, then the line will not have a mini-diamond (see figure in the row above).		Control Flow (Activity Edge with Guard) 
Default flow	For Data-Based Exclusive Decisions, one type of flow is the Default condition flow. This flow will be used only if all the other outgoing conditional flow is not true at runtime. These Sequence Flow will have a diagonal slash will be added to the beginning of the line (see the figure to the right). Note that it is an Open Issue whether Default Conditions will be used for Inclusive Decision situations.		Control Flow (Activity Edge with Guard set to “else”) 

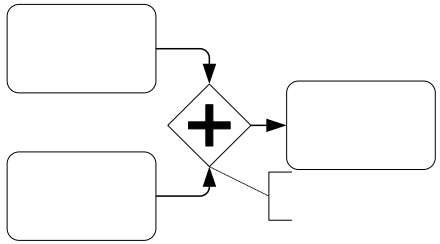
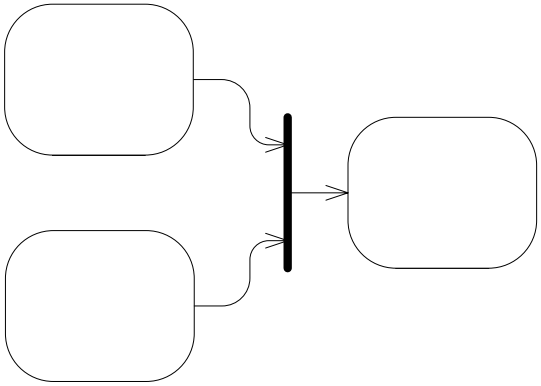
Exception flow	Exception flow occurs outside the normal flow of the Process and is based upon an Intermediate Event that occurs during the performance of the Process. All the Intermediate Event Triggers, except Compensation, can create Exception flow (see Message Trigger, Timer Trigger, Exception Trigger, Cancel Trigger, Rule Trigger, and Link Trigger)		This maps to an Interruptible Region. The details depends on the type of Intermediate Event Trigger. See the mappings shown in the different Trigger types as referenced in the second column. The example shown below is for an Exception Trigger. An interruptible region is created to surround the activity. Within the region, a signal receipt is added to catch the signal that was created elsewhere. An interrupting edge that exits the signal is used to exit the region (and the activity) when the signal arrives. 

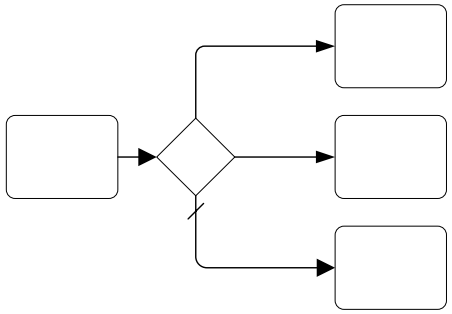
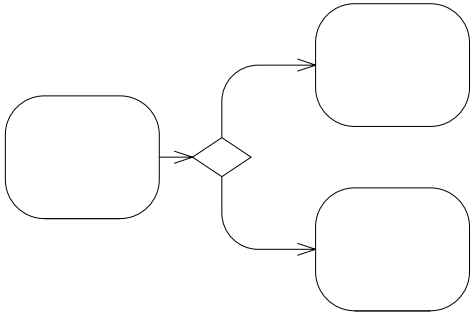
Message Flow	A Message Flow is used to show the flow of messages between two entities that are prepared to send and receive them. In BPMN, two separate Pools in the Diagram will represent the two entities.		Information Flow to an entity. This will be paired with BPMN pools that will be mapped to partitions and entities. 
Object Flow	In BPMN there is no specific creation of "Object Flow." However, the data/document input and output requirements are shown through Data Object Artifacts and Associations. This creates a complete decoupling of data and control flow. However, UML object flow MUST follow control flow. This creates some BPMN to UML mapping issues (see Ad-Hoc Process)	 Note that Sequence Flow is not required between the two activates in BPMN to show the flow of the object. 	This maps to Object Flow in both forms.  Note that a control flow is always paired with object flow.

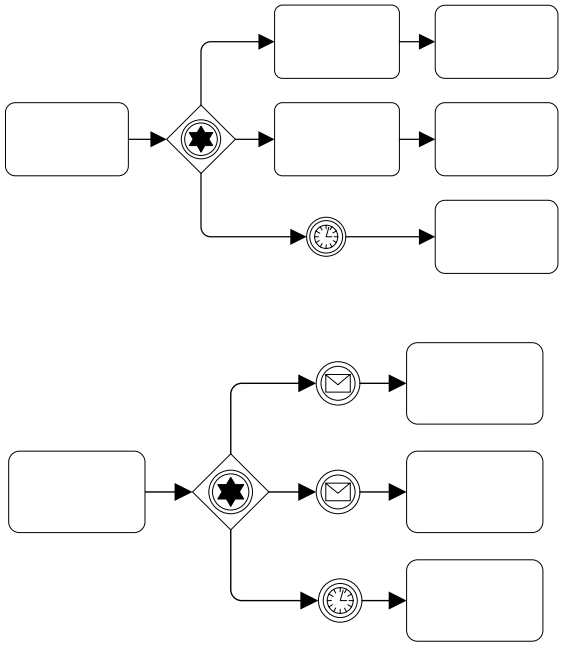
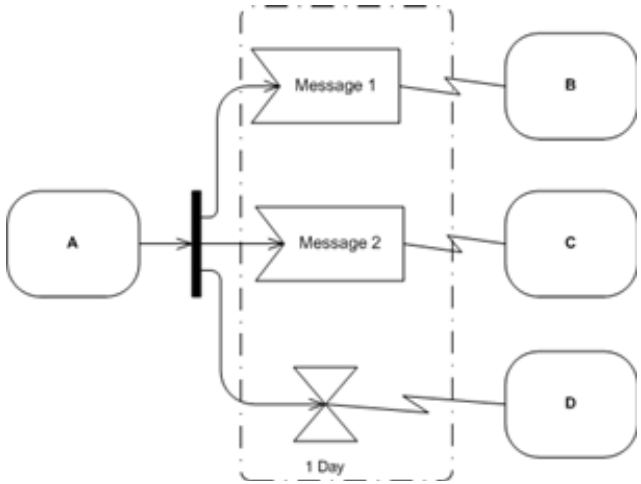
Complex Input/Output Requirements		This information is supported by activity Attributes. 	This maps to Parameter Sets attached to the activity to show AND and OR input/output requirements. 
Streaming Input/Output Requirements		The current version of BPMN does not graphically distinguish between types of inputs and outputs. This information is supported by activity Attributes. 	This maps to streaming style object flow or streaming pins. 

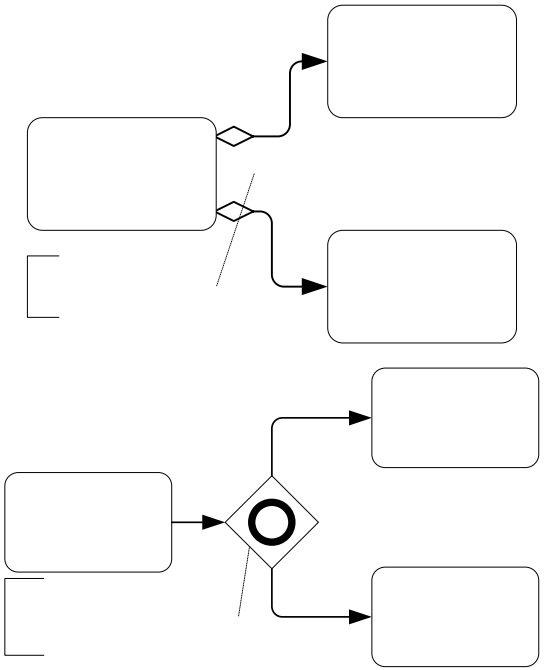
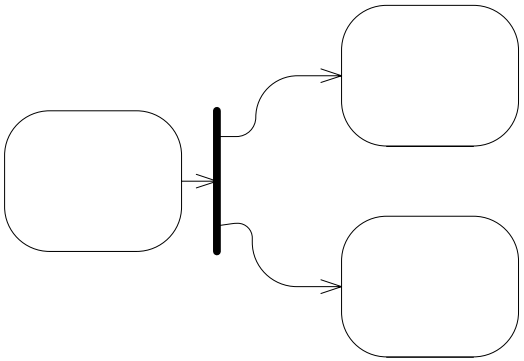
Compensation Association	Compensation Association occurs outside the normal flow of the Process and is based upon an event (a Cancel Intermediate Event) that is triggered through the failure of a Transaction or a Compensate Event. The target of the Association must be marked as a Compensation Activity.		There is no direct association with the original activity and the activity that is needed for compensation. Thus, the original activity is modeled by itself.  The activity used for compensation is set aside in a separate Process that is only activated if the Compensate signal is sent through some part of the original process. 
Data Object	Data Objects are considered artifacts because they do not have any direct affect on the Sequence Flow or Message Flow of the Process, but they do provide information about what the Process does.		This maps to an ObjectNode 

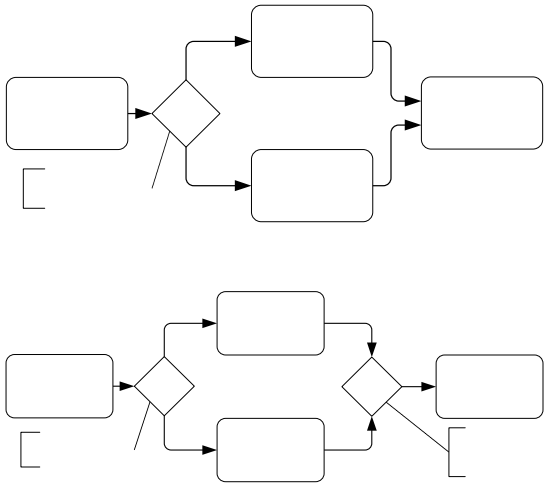
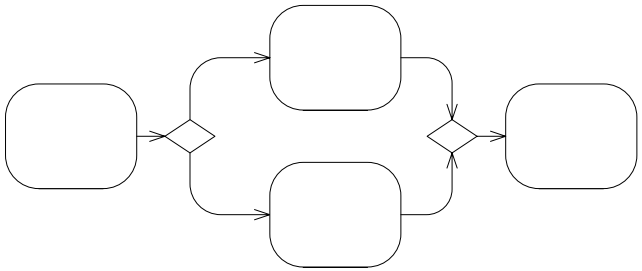
Database	Databases are considered artifacts because they do not have any direct affect on the Sequence Flow or Message Flow of the Process, but they do provide information about what the Process does.	This object is currently an extension to BPMN (through the extensibility of BPMN Artifacts). 	This maps to a DataStoreNode 
Fork (AND-Split)	BPMN uses the term "fork" to refer to the dividing of a path into two or more parallel paths (also known as an AND-Split). It is a place in the Process where activities can be performed concurrently, rather than serially. There are two options: Multiple Outgoing Sequence Flow can be used (see figure top-right). This represents "uncontrolled" flow is the preferred method for most situations. A Parallel (AND) Gateway can be used (see figure bottom-right). This will be used rarely, usually in combination with other Gateways.		This maps to a fork node. 

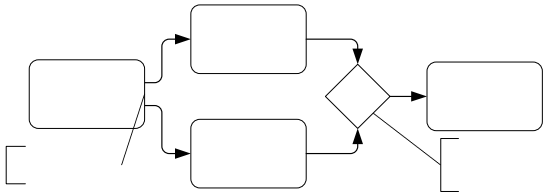
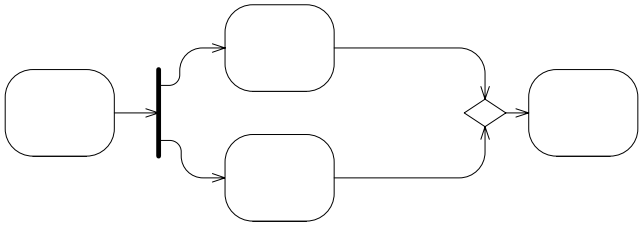
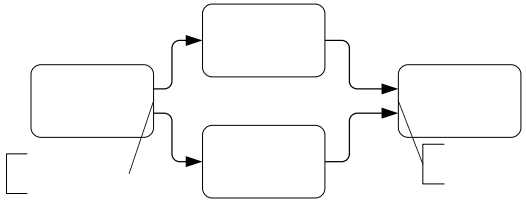
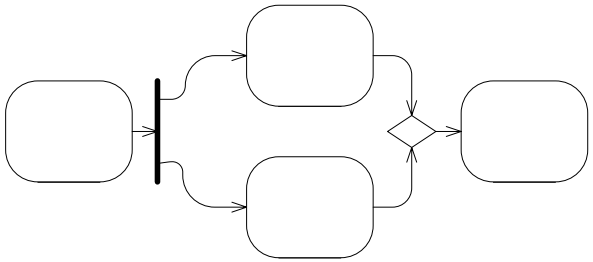
Join (AND-Join)	BPMN uses the term "join" to refer to the combining of two or more parallel paths into one path (also known as an AND-Join or synchronization). A Parallel (AND) Gateway is used to show the joining of multiple flows.		This maps to a Join Node 
Decision, Branching Point; (OR-Split)	Decisions are Gateways within a business process where the flow of control can take one or more alternative paths.	See next five rows.	
Exclusive	An Exclusive Gateway (XOR) restricts the flow such that only one of a set of alternatives may be chosen during runtime. There are two types of Exclusive Gateways: Data-based and Event-based.	See next two rows	See next two rows

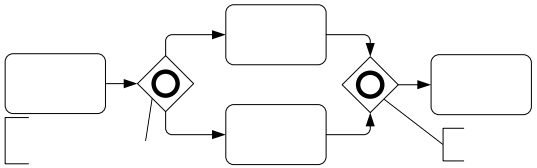
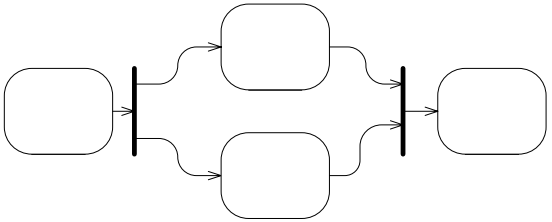
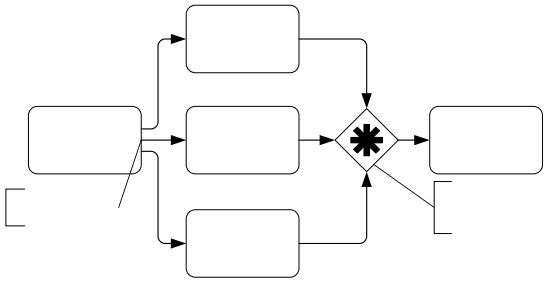
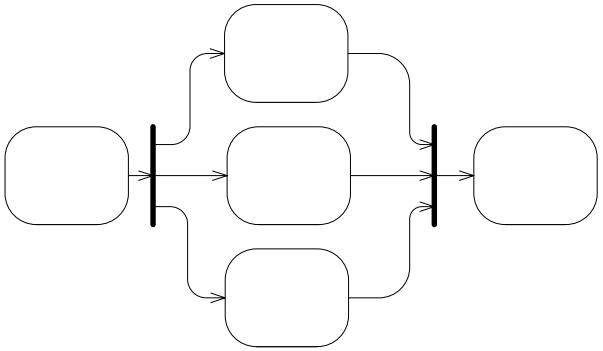
Data-Based	This Decision represents a branching point where Alternatives are based on conditional expressions contained within the outgoing Sequence Flow. Only one of the Alternatives will be chosen.		

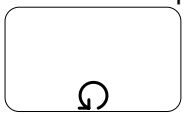
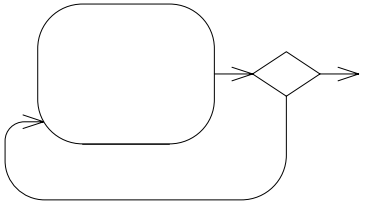
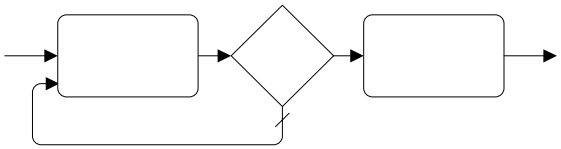
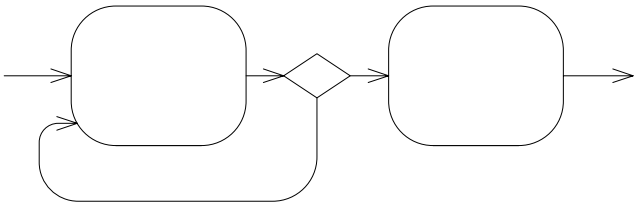
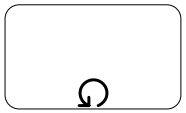
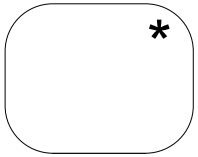
Event-Based	This Decision represents a branching point where Alternatives are based on an Event that occurs at that point in the Process. The specific Event, usually the receipt of a Message, determines which of the paths will be taken. Other types of Events can be used, such as Timer. Only one of the Alternatives will be chosen. There are two options for receiving Messages: Tasks of Type Receive can be used (see figure top-right). Intermediate Events of Type Message can be used (see figure bottom-right). The details of the mapping pattern will depend on the specific Intermediate Events that are used (see Message Trigger, Timer Trigger, Exception Trigger, Rule Trigger, and Link Trigger).		There is not a specific UML element to which this behavior will map, rather, it is a specific pattern of UML elements. In general, there will be a set of signals that will be used to determine the appropriate path to take. All the signals to be received by the pattern must be available at the same time, so they follow a fork node. But only one will be used—to the exclusion of all others. To do this, the signal receipts are all placed within an interruptible region. When a signal is received, this will lead to an interrupting edge that will cause the closing of the region, thus excluding the other signal receipts. 

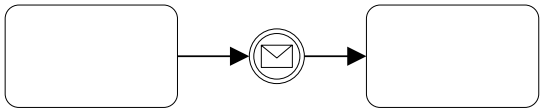
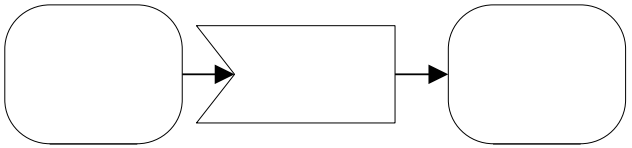
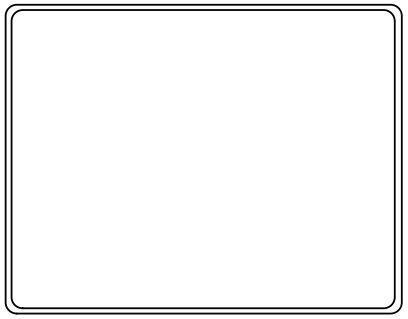
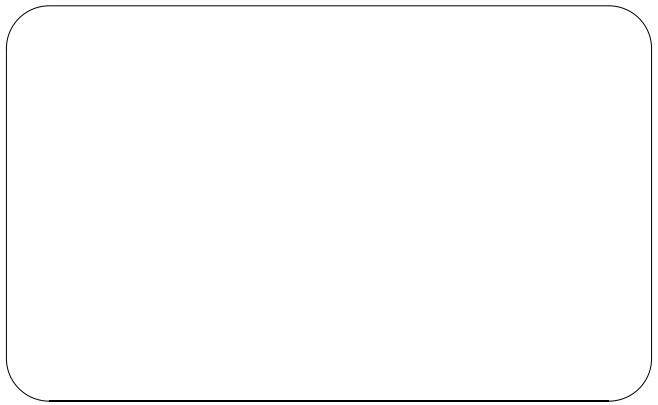
Inclusive	This Decision represents a branching point where Alternatives are based on conditional expressions contained within the outgoing Sequence Flow. In some sense it is a grouping of related independent Binary (Yes/No) Decisions. Since each path is independent, all combinations of the paths may be taken, from zero to all. However, it should be designed so that at least one path is taken. There are two versions of this type of Decision: The first uses a collection of conditional Sequence Flow, marked with mini-diamonds (see top-right figure). The second uses an OR Gateway, usually in combination with other Gateways (see bottom-right picture).		This maps to a fork node with outgoing control flow with guards. 

Merging (OR-Join)		See next five rows…	See next five rows…
Simple Merge			This maps to a merge node. 

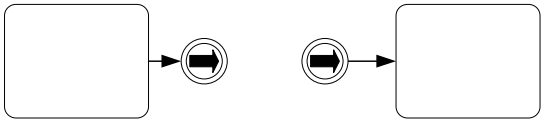
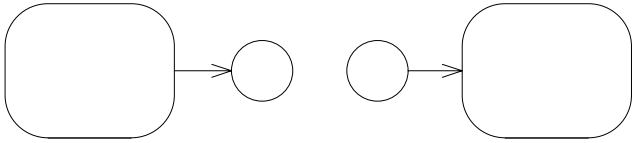
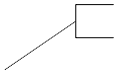
Discriminator	This creates a situation were there are multiple Tokens merging, but only the first Token is allowed to continue.	The Exclusive Gateway will accept the first Token and then exclude any others. 	This maps to a merge node. However, the edges leading to the merge node have guards that prohibit the flow if the other edge(s) leading to the same merge node have not yet occurred. Thus, the exclusive behavior is handled by the guards. 
Multiple Merge	This allows multiple Tokens to pass through the merge.	Uncontrolled (no Gateway) Sequence Flow into Task “D” will result in a separate instance of the Task for each Token that arrives. 	This maps to a merge node that is preceded (upstream) by a fork node. 

Synchronizing Merge	This synchs up the appropriate number of Tokens.	Either one or two Tokens will be merged. Activity “D” will occur once. 	This maps to a join node with a condition. 
N out of M Join	This is a join situation where a number (greater than 1), but less than the total number of Tokens is required for the process to continue. A condition can be specified, which will determine the appropriate number.	A Complex Gateway can be used to define the conditions required at the join. 	This maps to a condition for a Join node. 
Looping	BPMN provides 2 (two) mechanisms for looping within a Process.	See Next Two Figures	

Activity Looping	The properties of Tasks and Sub-Processes will determine if they are repeated or performed once. There are two types of loops: Standard and Multi-Instance. A small looping indicator will be displayed at the bottom-center of the activity.	Standard Loop 	The condition within the activity is extracted and a Decision is created. Then flow is connected back to the activity for the false path out of the Decision. 
Flow Looping	Loops can be created by connecting a Sequence Flow to an "upstream" object. An object is considered to be upstream if that object has an outgoing Sequence Flow that leads to a series of other Sequence Flows, the last of which is an incoming Sequence Flow for the original object.	Loop with Sequence Flow 	This maps to a Loop with control flow (or data flow). 
Multiple Instances	The attributes of Tasks and Sub-Processes will determine if they are repeated or performed once. A small parallel indicator will be displayed at the bottom-center of the activity.	Multiple Instance in Parallel  Multiple Instances in Sequence 	This maps to an Expansion Region. 

Process Break (something out of the control of the process makes the process pause)	A Process Break is a location in the Process that shows where an expected delay will occur within a Process. An Intermediate Event is used to show the actual behavior (see top-right figure). In addition, a Process Break artifact, as designed by a modeler or modeling tool, can be associated with the Event to highlight the location of the delay within the flow.		This maps to a Signal that occurs during the middle of the process. 
Transaction	A transaction is a Sub-Process that is supported by special protocol that insures that all parties involved have complete agreement that the activity should be completed or cancelled. The attributes of the activity will determine if the activity is a transaction. A double-lined boundary indicates that the activity is a Transaction.		Activity stereotyped as Transaction. [We could add the double-lined boundary as part of the stereotype.] 

Nested Sub-Process (Inline Block)	A nested Sub-Process is an activity that shares the same set of data as its parent process. This is opposed to a Sub-Process that is independent, re-usable, and referenced from the parent process. Data needs to be passed to the referenced Sub-Process, but not to the nested Sub-Process.	There is no special indicator for nested Sub-Processes, it is supported by Sub-Process attributes.	This maps to a StructuredActivityNode
Group (a box around a group of objects for documentation purposes)	A grouping of activities that does not affect the Sequence Flow. The grouping can be used for documentation or analysis purposes. Groups can also be used to identify the activities of a distributed transaction that is shown across Pools.		Interruptible Region (without an Interruption Edge)? Partition 
Off-Page Connector	Generally used for printing, this object will show where the Sequence Flow leaves one page and then restarts on the next page. A Link Intermediate Event can be used as an Off-Page Connector.		This maps to an Activity Edge Connector 

Go To object	Related to Off-Page Connector. Creates a virtual connection between two locations of a diagram.	The exact use of Link Intermediate Events is an open issue for BPMN. 	This maps to an Activity Edge Connector 
Association	An Association is used to associate information with flow objects. Text and graphical non-flow objects can be associated with the flow objects.		Not sure. Could be related to data flow, but also relates to other Artifacts besides Data Objects. Data Flow
Text Annotation (attached with an Association)	Text Annotations are a mechanism for a modeler to provide additional information for the reader of a BPMN Diagram.		This maps to a Note 
Pool	A Pool is a "swimlane" and a graphical container for partitioning a set of activities from other Pools, usually in the context of B2B situations.		Partition with no parent partition. It is also an Entity so that Information Flow can be shown between two Pools. 

Lanes	A Lane is a sub-partition within a Pool and will extend the entire length of the Pool, either vertically or horizontally. Lanes are used to organize and categorize activities within a Pool.		This maps to a Partition. 

Table 2 BPD Complete Element Set

Examples & Discussion

Comment: Steve White to provide some examples.