Business Process Modeling Notation Specification

This OMG document replaces the final adopted specification (dtc/07-07-0X). It is an OMG XXXXX Specification, which has been approved by the OMG board and technical plenaries, and is currently in the XXXXX phase. Comments on the content of this document are welcomed, and should be directed to *issues@omg.org* by Month, DD 2007.

You may view the pending issues for this specification from the OMG revision issues web page *http://www.omg.org/issues/*; however, at the time of this writing there were no pending issues.

Business Process Modeling Notation (BPMN) Specification

Version 1.1 --- DRAFT
dtc/07-07-XX

OBJECT MANAGEMENT GROUP

IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 250 First Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

The OMG Object Management Group Logo®, CORBA®, CORBA Academy®, The Information Brokerage®, XMI® and IIOP® are registered trademarks of the Object Management Group. OMG™, Object Management Group™, CORBA logos™, OMG Interface Definition Language (IDL)™, The Architecture of Choice for a Changing World™, CORBAservices™, CORBAfacilities™, CORBAmed™, CORBAnet™, Integrate 2002™, Middleware That's Everywhere™, UML™, Unified Modeling Language™, The UML Cube logo™, MOF™, CWM™, The CWM Logo™, Model Driven Architecture™, Model Driven Architecture Logos™, MDA™, OMG Model Driven Architecture™, OMG MDA™ and the XMI Logo™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents, Report a Bug/Issue (http://www.omg.org/technology/agreement.htm).

Table of Contents

List of Figures

List of Tables

List of Examples

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A catalog of all OMG Specifications Catalog is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications.

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM).

Platform Specific Model and Interface Specifications

- CORBAservices

- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications.

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. (as of January 16, 2006) at:

> OMG Headquarters
> 140 Kendrick Street
> Building A, Suite 300
> Needham, MA 02494
> USA
> Tel: +1-781-444-0404
> Fax: +1-781-444-0320
> Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/technology/agreement.htm*.

1 Scope

The **Business Process Management Initiative** (BPMI) has developed a standard **Business Process Modeling Notation** (BPMN). The primary goal of BPMN is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation.

Another goal, but no less important, is to ensure that XML languages designed for the execution of business processes, such as **BPEL4WS** (Business Process Execution Language for Web Services), can be visualized with a business-oriented notation.

This specification defines the notation and semantics of a **Business Process Diagram** (BPD) and represents the amalgamation of best practices within the business modeling community. The intent of BPMN is to standardize a business process modeling notation in the face of many different modeling notations and viewpoints. In doing so, BPMN will provide a simple means of communicating process information to other business users, process implementers, customers, and suppliers.

 This version of the specification does not specify a mechanism for exchange of BPMN diagrams.

This version of the specification does not specify a mechanism for the exchange of the semantic model of a process depicted by a BPMN diagram.

Note – Exchange of models of BPMN process semantics and diagrams is the subject of other ongoing standards activities.

This version of the specification does not specify a normative mapping from BPMN to WSBPEL.

Note – This version does provide a non-normative mapping from BPMN to WSBPEL, but the BPMN specification itself is known to be incomplete with respect to capturing all the required information for WSBPEL. So the mapping is insufficient, in any case.

The membership of the BPMI Notation Working Group has brought forth expertise and experience with many existing notations and has sought to consolidate the best ideas from these divergent notations into a single standard notation. Examples of other notations or methodologies that were reviewed are UML Activity Diagram, UML EDOC Business Processes, IDEF, ebXML BPSS, Activity-Decision Flow (ADF) Diagram, RosettaNet, LOVeM, and Event-Process Chains (EPCs).

2 Conformance

An implementation claiming conformance to this specification shall comply with all of the requirements set forth in subclauses 2.1, 2.2 and 2.3 below.

2.1 Visual Appearance

A key element of BPMN is the choice of shapes and icons used for the graphical elements identified in this specification. The intent is to create a standard visual language that all process modelers will recognize and understand.

An implementation that creates and displays BPMN Diagrams shall use the graphical elements, shapes and markers specified in clauses 9-10 as the diagrammatic elements that represent the specified concepts.

Note – There is flexibility in the size, color, line style, and text positions of the defined graphical elements, except where otherwise specified.

The following extensions to a BPMN Diagram are permitted:

- New markers or indicators MAY be added to the specified graphical elements. These markers or indicators could be used to highlight a specific attribute of a BPMN element or to represent a new subtype of the corresponding concept. (See also 2.4 below)

- A new shape representing a kind of Artifact may be added to a Diagram, but the new Artifact shape SHALL NOT conflict with the shape specified for any other BPMN object or marker.

- Graphical elements may be colored, and the coloring may have specified semantics that extend the information conveyed by the element as specified in this standard.

- The line style of a graphical element may be changed, but that change SHALL NOT conflict with any other line style required by this specification.

An extension SHALL NOT change the specified shape of a defined graphical element or marker. (e.g., changing a square into a triangle, or changing rounded corners into squared corners, etc.).

2.2   Structural Conformance

An implementation that creates and displays BPMN diagrams shall conform to the specifications and restrictions in Clauses 8-10 with respect to the connections and other diagrammatic relationships between graphical elements. Where permitted or required connections are specified as conditional and based on attributes of the corresponding concepts, the implementation shall ensure the correspondence between the connections and the values of those attributes.

Note – In general, these connections and relationships have specified semantic interpretations, which specify interactions among the process concepts represented by the graphical elements. Conditional relationships based on attributes represent specific variations in behavior. Structural conformance therefore guarantees the correct interpretation of the diagram as a specification of process, in terms of flows of control and information.

Throughout the document, structural specifications will be appear in paragraphs using a special shaped bullet:

Example:

- ◆ A Task MAY be a target for Sequence Flow; it can have multiple incoming Flows. An Incoming Flow MAY be from an alternative path and/or a parallel paths.

2.3   Semantic Elements

This specification defines many semantic concepts used in defining processes, and associates them with graphical elements, markers, and connections. To the extent that an implementation provides an interpretation of the BPMN diagram as a semantic specification of process, the interpretation shall be consistent with the semantic interpretation herein specified.

Note – The intent here is that a BPMN diagram used as a "workflow specification" will have the interpretation specified in

this standard, somewhat extended or narrowed by the characteristics of the workflow system. Similarly, when a BPMN diagram used as a specification for the processes and interactions of software agents, any generated software will reflect the semantics of the diagram as specified in this standard, possibly narrowed or extended by the characteristics of the software implementation.

2.4   Attributes and Properties

This specification defines a number of attributes and properties of the semantic objects represented by the graphical elements, markers, and connections. Some of these attributes are purely representational and are so marked, and some have required representations. Some attributes are specified as mandatory, but have no representation or only optional representation. And some attributes are specified as optional.

For every attribute or property that is specified as mandatory, a conforming implementation SHALL provide some mechanism by which values of that attribute or property can be created and displayed. This mechanism SHALL permit the user to create or view these values for each BPMN object specified to have that attribute or property.

Where a graphical representation for that attribute or property is specified as required, that graphical representation SHALL be used.

Where a graphical representation for that attribute or property is specified as optional, the implementation MAY use either a graphical representation or some other mechanism. If a graphical representation is used, it SHALL be the representation specified.

Where no graphical representation for that attribute or property is specified, the implementation MAY use either a graphical representation or some other mechanism. If a graphical representation is used, it SHALL NOT conflict with the specified graphical representation of any other BPMN object.

2.5   Extended and Optional Elements

A conforming implementation is not required to support any element or attribute that is specified herein to be non-normative or informative.

In each instance in which this specification defines a feature to be "optional", it specifies whether the option is in:

- how the feature shall be displayed,
- whether the feature shall be displayed
- whether the feature shall be supported.

A conforming implementation is not required to support any feature whose support is specified to be optional. If an implementation supports an optional feature, it SHALL support it as specified.

A conforming implementation SHALL support any "optional" feature for which the option is only in whether or how it shall be displayed.

3   Normative References

3.1   Normative

RFC-2119

- Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, IETF RFC 2119, March 1997
  http://www.ietf.org/rfc/rfc2119.txt

3.2   Non-Normative

Activity Service

- Additional Structuring Mechanism for the OTS specification, OMG, June 1999
  http://www.omg.org

- J2EE Activity Service for Extended Transactions (JSR 95), JCP
  http://www.jcp.org/jsr/detail/95.jsp

BPEL4WS

- (BPEL4WS) 1.1, IBM/Microsoft/BEA/SAP/Siebel, July 2002
  http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

Business Process Definition

- Final Response to OMG BPD RFP, OMG, March 2007, bmi/07-03-01
  http://www.omg.org

Business Process Modeling

- Jean-Jacques Dubray, "A Novel Approach for Modeling Business Process Definitions," 2002
  http://www.ebpml.org/ebpml2.2.doc

Business Transaction Protocol

- OASIS BTP Technical Committee, June, 2002
  http://www.oasis-open.org/committees/download.php/1184/2002-06-03.BTP_cttee_spec_1.0.pdf

BPML

- (BPML) 1.0, BPMI, January 2003
  http://www.BPMI.org

Dublin Core Meta Data

- Dublin Core Metadata Element Set, Dublin Core Metadata Initiative
  http://dublincore.org/documents/dces/

ebXML BPSS

- Jean-Jacques Dubray, "A new model for ebXML BPSS Multi-party Collaborations and Web Services Choreography,"

2002
http://www.ebpml.org/ebpml.doc

OMG UML

- Unified Modeling Language Specification V2.1.1: Superstructure, OMG, February 2007, formal/2007-02-05
  http://www.omg.org

Open Nested Transactions

- Concepts and Applications of Multilevel Transactions and Open Nested Transactions, Gerhard Weikum, Hans-J. Schek, 1992
  http://citeseer.nj.nec.com/weikum92concepts.html

RDF

- RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft
  http://www.w3.org/TR/rdf-schema/

SOAP 1.2

- SOAP Version 1.2 Part 1: Messaging Framework, W3C Working Draft
  http://www.w3.org/TR/soap12-part1/

- SOAP Version 1.2 Part21: Adjuncts, W3C Working Draft
  http://www.w3.org/TR/soap12-part2/

UDDI

- Universal Description, Discovery and Integration, Ariba, IBM and Microsoft, UDDI.org.
  http://www.uddi.org

URI

- Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter, IETF RFC 2396, August 1998
  http://www.ietf.org/rfc/rfc2396.txt

WfMC Glossary

- Workflow Management Coalition Terminology and Glossary.
  http://www.wfmc.org/standards/docs.htm

Web Services Transaction

- (WS-Transaction) 1.0, IBM/Microsoft/BEA, August, 2002
  http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/

WSBPEL

- Web Services Business Process Execution Language (WSBPEL) 2.0, Committee Specification, January 2007
  http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.pdf

WSDL

- Web Services Description Language (WSDL) 2.0, W3C Proposed Recommendation, May 2007
  http://www.w3.org/TR/2007/PR-wsdl20-20070523/l

XML 1.0 (Second Edition)

- Extensible Markup Language (XML) 1.0, Second Edition, Tim Bray et al., eds., W3C, 6 October 2000
  http://www.w3.org/TR/REC-xml

XML-Namespaces

- Namespaces in XML, Tim Bray et al., eds., W3C, 14 January 1999
  http://www.w3.org/TR/REC-xml-names

XML-Schema

- XML Schema Part 1: Structures, Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, W3C, 2 May 2001
  http://www.w3.org/TR/xmlschema-1//

- XML Schema Part 2: Datatypes, Paul V. Biron and Ashok Malhotra, eds., W3C, 2 May 2001
  http://www.w3.org/TR/xmlschema-2/

XPath

- XML Path Language (XPath) 1.0, James Clark and Steve DeRose, eds., W3C, 16 November 1999
  http://www.w3.org/TR/xpath

XPDL

- Workflow Management Coalition XML Process Definition Language, version 2.0.
  http://www.wfmc.org/standards/docs.htm

4 Terms and Definitions

See Annex C - Glossary.

5 Symbols

There are no symbols defined in this specification.

6   Additional Information

6.1   Conventions

The section introduces the conventions used in this document. This includes (text) notational conventions and notations for schema components. Also included are designated namespace definitions.

6.1.1   Typographical and Linguistic Conventions and Style

This specification incorporates the following conventions:

- The keywords "MUST," "MUST NOT," "REQUIRED," "SHALL," "MUST NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in this document are to be interpreted as described in RFC-2119.

- A **term** is a word or phrase that has a special meaning. When a term is defined, the term name is highlighted in **bold** typeface.

- A reference to another definition, section, or specification is highlighted with <u>underlined</u> typeface and provides a link to the relevant location in this specification.

- A reference to an element, attribute, or BPMN construct is highlighted with a capitalized word (e.g., Sub-Process).

- A reference to a BPEL4WS element, attribute, or construct is highlighted with an italic lower-case word, usually preceded by the word "BPEL4WS" (e.g., BPEL4WS *pick*).

- Non-normative examples are set off in boxes and accompanied by a brief explanation.

- XML and pseudo code is highlighted with `mono-spaced` typeface. Different font colors may be used to highlight the different components of the XML code.

- The cardinality of any content part is specified using the following operators:
    - \<none\> — exactly once
    - (0-1) — 0 or 1
    - (0-n) — 0 or more
    - (1-n) — 1 or more

- Attributes separated by | and grouped within (and) — alternative values
    - \<value\> — default value
    - \<type\> — the type of the attribute

6.1.2 Abbreviations

The following abbreviations may be used throughout this document:

This abbreviation	Refers to
BPEL4WS	Business Process Execution Language for Web Services (see BPEL4WS). This abbreviation refers specifically to version 1.1 of the specification.
WSDL	Web Service Description Language (see WSDL). This abbreviation refers specifically to the W3C Technical Note, 15 March 2001, but is intended to support future versions of the WSDL specification

6.2   Structure of this Document

The BPMN specification defines the Business Process Diagram modeling objects, their semantics, their mapping to BPEL4WS, and is comprised of the following topics:

BPMN Overview provides an introduction to BPMN, its requirements, and discusses the range of modeling purposes that BPMN can convey.

Business Process Diagrams provides a summary of the BPMN graphical elements and their relationships.

Business Process Diagram Graphical Objects details the graphical representation, attributes, and semantics of the behavior of BPMN Diagram elements.

Business Process Diagram Connecting Objects defines the graphical objects used to connect two objects together (i.e., the connecting lines of the Diagram) and how flow progresses through a Process (i.e., through a straight sequence or through the creation of parallel or alternative paths).

BPMN by Example provides a walkthrough of a sample Process using BPMN.

Annex A: Mapping to BPEL4WS provides a mechanism for converting a Business Process to a BPEL4WS document, provides and example of Process mapping, and provides a full sample of BPEL4WS code based on the example process mapping.

Annex B: BPMN Element Attributes and Types provides the complete set of BPMN Element attributes, which are first presented in Chapters 8, 9, and 10, and the definition of types that support the attributes.

Annex C: Glossary presents an alphabetical index of terms that are relevant to practitioners of BPMN.

6.3   Acknowledgements

The following companies submitted and/or supported parts of this specification:

- 88Solutions

- Adobe

- Adaptive

- Appian

- Axway Software

- BEA

- BizAgi

- Boeing

- Borland

- BPM Focus

- Business Rules Group

- Casewise

- Computas

- EDS

- Embarcadero Technologies

- Fair, Isaac & Company

- Global 360

- Graham Technology

- Hewlett-Packard

- IBM Corporation

- IBM Corporation (FileNet)

- Infosys

- iGrafx

- Intalio

- International Performance Group

- ITPearls AG

- KnowGravity

- Lombardi Software

- Mega International

- NIST

- No Magic, Inc.

- oose Innovative Informatik GmbH

- Pegasystems

- Proforma

- Sandpiper Software

- SAP

- Software AG (webMethods)

- Sterling Commerce

- Sun

- Sun (SeeBeyond Technology Corporation)

- Sybase

- Tall Tree Labs

- Telelogic (Popkin Software)

- Tibco

- Troux Technologies

- Unisys

- U.S. Department of Treasury

The following person was the main author/editor of the specification: Stephen A White. The following persons were members of the core teams that contributed to the content specification: Michael Anthony, Assaf Arkin, Sylvan Astier, Rob Bartel, Ed Barkmeyer, Conrad Bock, Donna Burbank, Steinar Carlsen, Petko Chobantonov, Ugo Corda, Fred Cummins, Bob Daniel, Tony Fletcher, Steven Forgey, Karl Frank, Jean-Luc Giraud, Brian James, George Keeling, Markus Klink, Antoine Lonjon, Monica Martin, Lee Mason, Frank McCabe, Dale Moberg, Martin Owen, Pete Rivett, Suzette Samoojh, Jesus Sanchez, Robert Shapiro, Bob Smith, Manfred Sturm, Balasubramanian (Bala) Suryanarayanan, Michelle Vanchu-Orozco, David Williams, and Paul Wuethrich.

In addition, the following persons contributed valuable ideas and feedback that improved the content and the quality of this specification: Ashish Agrawal, Mike Amend, Don Baisley, Steve Ball, Pranab Baruah, Olivier Bigard, Justin Brunt, Cory Casanave, Pam Corsini, Bernard Debauche, Joachim (Jim) Frank, David Frankel, John Hall, Paul Harmon, Damion Heredia, Cyril Jaoen, Jana Koehler, Manfred Koethe, Jochen Kuester, Philip Larson, Mike Marin, Derek Miers, Alex Moffat, Jishnu Mukerji, Roberta Norin, Jog Raj, James Rubert, Markus Schacher, Ed Seidewitz, James Taylor, Bobbin Teegarden, Roy Thompson, Paul Vincent, Peter Walker, and Tim Weilkiens.

7   Overview

There has been much activity in the past few years in developing web service-based XML execution languages for Business Process Management (BPM) systems. Languages such as BPEL4WS provide a formal mechanism for the definition of business processes. The key element of such languages is that they are optimized for the operation and inter-operation of BPM Systems. The optimization of these languages for software operations renders them less suited for direct use by humans to design, manage, and monitor business processes. BPEL4WS has both graph and block structures and utilizes the principles of formal mathematical models, such as pi-calculus[1]. This technical underpinning provides the foundation for business process execution to handle the complex nature of both internal and B2B interactions and take advantage of the benefits of Web services. Given the nature of BPEL4WS, a complex business process could be organized in a potentially complex, disjointed, and unintuitive format that is handled very well by a software system (or a computer programmer), but would be hard to understand by the business analysts and managers tasked to develop, manage, and monitor the process. Thus, there is a human level of "inter-operability" or "portability" that is not addressed by these web service-based XML execution languages.

Business people are very comfortable with visualizing business processes in a flow-chart format. There are thousands of business analysts studying the way companies work and defining business processes with simple flow charts. This creates a technical gap between the format of the initial design of business processes and the format of the languages, such as BPEL4WS, that will execute these business processes. This gap needs to be bridged with a formal mechanism that maps the appropriate visualization of the business processes (a notation) to the appropriate execution format (a BPM execution language) for these business processes.

Inter-operation of business processes at the human level, rather than the software engine level, can be solved with standardization of the Business Process Modeling Notation (BPMN). BPMN provides a Business Process Diagram (BPD), which is a Diagram designed for use by the people who design and manage business processes. BPMN also provides a mapping to an execution language of BPM Systems (BPEL4WS). Thus, BPMN would provide a standard visualization mechanism for business processes defined in an execution optimized business process language.

BPMN will provide businesses with the capability of understanding their internal business procedures in a graphical notation and will give organizations the ability to communicate these procedures in a standard manner. Currently, there are scores of process modeling tools and methodologies. Given that individuals will move from one company to another and that companies will merge and diverge, it is likely that business analysts are required to understand multiple representations of business processes--potentially different representations of the same process as it moves through its lifecycle of development, implementation, execution, monitoring, and analysis. Therefore, a standard graphical notation will facilitate the understanding of the performance collaborations and business transactions within and between the organizations. This will ensure that businesses will understand themselves and participants in their business and will enable organizations to adjust to new internal and B2B business circumstances quickly. To do this, BPMN will follow the tradition of flowcharting notations for readability; yet still provide a mapping to the executable constructs. BPMI is using the experience of the business process notations that have preceded BPMN to create the next generation notation that combines readability, flexibility, and expandability.

BPMN will also advance the capabilities of traditional business process notations by inherently handling B2B business process concepts, such as public and private processes and choreographies, as well as advanced modeling concepts, such as exception handling, transactions, and compensation.

---

1.   See Milner, 1999, "Communicating and Mobile Systems: the Π-Calculus," Cambridge University Press. ISBN 0 521 64320 1 (hc.) ISBN 0 521 65869 1 (pbk.)

7.1 BPMN Scope

BPMN will be constrained to support only the concepts of modeling that are applicable to business processes. This means that other types of modeling done by organizations for business purposes will be out of scope for BPMN. For example, the modeling of the following will not be a part of BPMN:

- Organizational structures and Resources

- Functional breakdowns

- Data and information models

- Strategy

- Business Rules

Since these types of high-level modeling either directly or indirectly affects business processes, the relationships between BPMN and other high-level business modeling will be defined more formally as BPMN and other specifications are advanced.

In addition, while BPMN will show the flow of data (messages), and the association of data Artifacts to activities, it is not a data flow Diagram.

7.1.1 Uses of BPMN

Business process modeling is used to communicate a wide variety of information to a wide variety of audiences. BPMN is designed to cover many types of modeling and allows the creation of end-to-end business processes. The structural elements of BPMN will allow the viewer to be able to easily differentiate between sections of a BPMN Diagram.

There are three basic types of sub-models within an end-to-end BPMN model:

- Private (internal) business processes

- Abstract (public) processes

- Collaboration (global) Processes

Note – The terminology used to describe the different types of processes has not been standardized. Definitions of these terms are in flux. There is work being done in the World Wide Web Consortium (W3C) and in the Organization for the Advancement of Structured Information Standards (OASIS) that will hopefully consolidate these terms.

Some BPMN specification terms regarding the use of Swimlanes (e.g., Pools and Lanes) are used in the descriptions below. Refer to "Swimlanes (Pools and Lanes)" on page 267 for more details on how these elements are used in a BPD.

Private (Internal) Business Processes

Private business processes are those internal to a specific organization and are the types of processes that have been generally called workflow or BPM processes (see Figure 7.1). A single private business process may be mapped to one or more BPEL4WS documents.

If Swimlanes are used then a private business process will be contained within a single Pool. The Sequence Flow of the Process is therefore contained within the Pool and cannot cross the boundaries of the Pool. Message Flow can cross the Pool boundary to show the interactions that exist between separate private business processes. Thus, a single Business Process Diagram may show multiple private business processes, each with separate mappings to BPEL4WS.
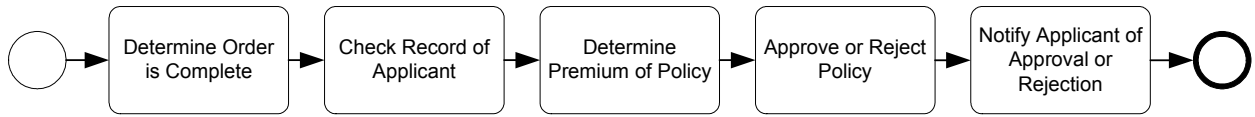
Figure 7.1 - Example of Private Business Process

Abstract (Public) Processes

This represents the interactions between a private business process and another process or participant (see Figure 7.2). Only those activities that are used to communicate outside the private business process, plus the appropriate flow control mechanisms, are included in the abstract process. All other "internal" activities of the private business process are not shown in the abstract process. Thus, the abstract process shows to the outside world the sequence of messages that are required to interact with that business process. A single abstract process may be mapped to a single BPEL4WS abstract *process* (however, this mapping will not be done in this version of the specification).

Abstract processes are contained within a Pool and can be modeled separately or within a larger BPMN Diagram to show the Message Flow between the abstract process activities and other entities. If the abstract process is in the same Diagram as its corresponding private business process, then the activities that are common to both processes can be associated.
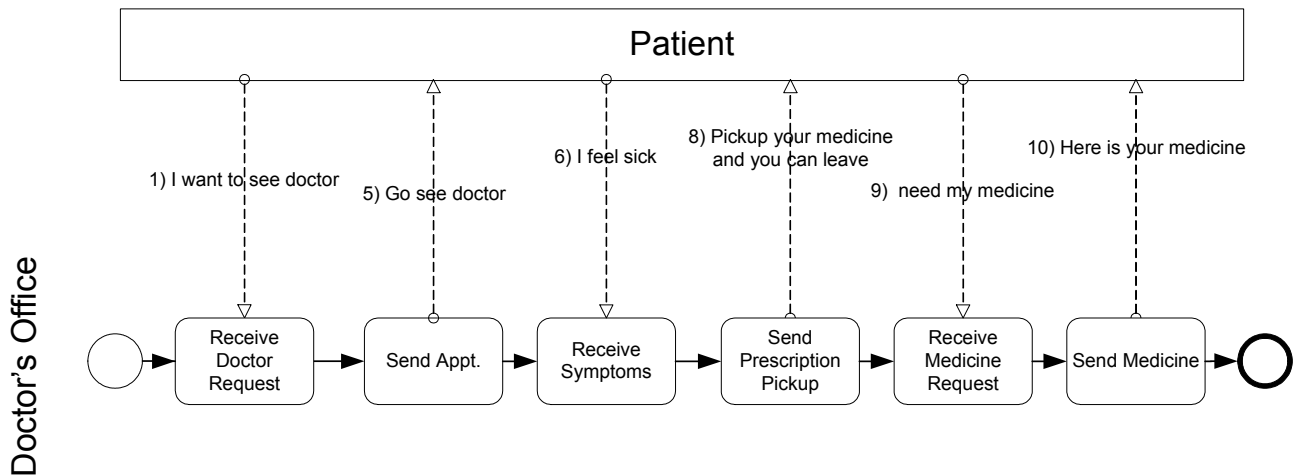
Figure 7.2 - Example of an Abstract Business Process

Collaboration (Global) Processes

A collaboration process depicts the interactions between two or more business entities. These interactions are defined as a sequence of activities that represent the message exchange patterns between the entities involved. A single collaboration process may be mapped to various collaboration languages, such as ebXML BPSS, RosettaNet, or the resultant specification from the W3C Choreography Working Group (however, these mappings are considered as future directions for BPMN).

The collaboration process can be shown as two or more abstract processes communicating with each other (see Figure 7.3). With an abstract process, the activities for the collaboration participants can be considered the "touch-points" between the participants. The actual (executable) processes are likely to have much more activity and detail than what is shown in the abstract processes.
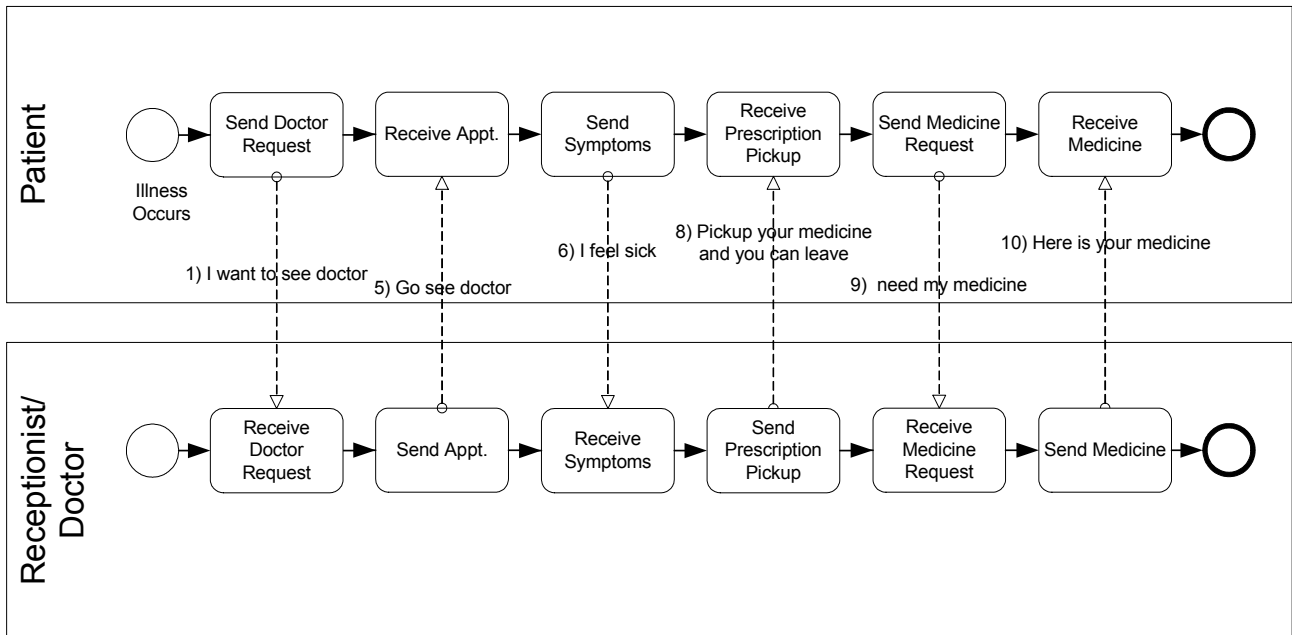
Figure 7.3 - Example of a Collaboration Business Process

Types of BPD Diagrams

Within and between these three BPMN sub-models, many types of Diagrams can be created. The following are the types of business processes that can be modeled with BPMN (those with asterisks may not map to an executable language):

- High-level private process activities (not functional breakdown)*
- Detailed private business process
    - As-is or old business process*
    - To-be or new business process
- Detailed private business process with interactions to one or more external entities (or "Black Box" processes)
- Two or more detailed private business processes interacting
- Detailed private business process relationship to Abstract Process
- Detailed private business process relationship to Collaboration Process
- Two or more Abstract Processes*
- Abstract Process relationship to Collaboration Process*
- Collaboration Process only (e.g., ebXML BPSS or RosettaNet)*
- Two or more detailed private business processes interacting through their Abstract Processes

- Two or more detailed private business processes interacting through a Collaboration Process

- Two or more detailed private business processes interacting through their Abstract Processes and a Collaboration Process

BPMN is designed to allow all the above types of Diagrams. However, it should be cautioned that if too many types of sub-models are combined, such as three or more private processes with message flow between each of them, then the Diagram may become too hard for someone to understand. Thus, we recommend that the modeler pick a focused purpose for the BPD, such as a private process, or a collaboration process.

BPMN mappings

Since BPMN covers such a wide range of usage, it will map to more than one lower-level specification language:

- BPEL4WS are the primary languages that BPMN will map to, but they only cover a single executable private business process. If a BPMN Diagram depicts more than one internal business process, then there will a separate mapping for each on the internal business processes.

- The abstract sections of a BPMN Diagram will be mapped to Web service interfaces specifications, such as the abstract processes of BPEL4WS.

- The Collaboration model sections of a BPMN may be mapped Collaboration models such as ebXML BPSS, Rosetta-Net, and the W3C Choreography Working Group Specification (when it is completed).

This specification will only cover a mapping to BPEL4WS. Mappings to other specifications will have to be a separate effort, or perhaps a future direction of BPMN (beyond Version 1.0 of the BPMN specification). It is hard to predict which mappings will be applied to BPMN at this point, since process language specifications is a volatile area of work, with many new offerings and mergings.

A BPD is not designed to graphically convey all the information required to execute a business process. Thus, the graphic elements of BPMN will be supported by attributes that will supply the additional information required to enable a mapping to BPEL4WS. A complete list of all the element attributes can be found in Annex B.

7.1.2   Diagram Point of View

Since a BPMN Diagram may depict the Processes of different Participants, each Participant may view the Diagram differently. That is, the Participants have different points of view regarding how the Processes will apply to them. Some of the activities will be internal to the Participant (meaning performed by or under control of the Participant) and other activities will be external to the Participant. Each Participant will have a different perspective as to which are internal and external. At runtime, the difference between internal and external activities is important in how a Participant can view the status of the activities or trouble-shoot any problems. However, the Diagram itself remains the same. Figure 7.3, above, displays a Business Process that has two points of view. One point of view is of a Patient, the other is of the Doctor's office. The Diagram shows the activities of both participants in the Process, but when the Process is actually being performed, each Participant will only have control over their own activities.

Although the Diagram point of view is important for a viewer of the Diagram to understand how the behavior of the Process will relate to that viewer, BPMN will not currently specify any graphical mechanisms to highlight the point of view. It is open to the modeler or modeling tool vendor to provide any visual cues to emphasize this characteristic of a Diagram.

7.1.3 Extensibility of BPMN and Vertical Domains

BPMN is intended to be extensible by modelers and modeling tools. This extensibility allows modelers to add non-standard elements or Artifacts to satisfy a specific need, such as the unique requirements of a vertical domain. While extensible, BPMN Diagrams should still have the basic look-and-feel so that a Diagram by any modeler should be easily understood by any viewer of the Diagram. Thus the footprint of the basic flow elements (Events, activities, and Gateways) should not be altered. Nor should any new flow elements be added to a BPD, since there is no specification as to how Sequence and Message Flow will connect to any new Flow Object. In addition, mappings to execution languages may be affected if new flow elements are added. To satisfy additional modeling concepts that are not part of the basic set of flow elements, BPMN provides the concept of Artifacts that can be linked to the existing Flow Objects through Associations. Thus, Artifacts do not affect the basic Sequence or Message Flow, nor do they affect mappings to execution languages.

The graphical elements of BPMN are designed to be open to allow specialized markers to convey specialized information. For example, the three types of Events all have open centers for the markers that BPMN standardizes as well as user-defined markers.

8   Business Process Diagrams

This chapter provides a summary of the BPMN graphical objects and their relationships. More details on the concepts will be provided in "Business Process Diagram Graphical Objects" on page 35 and "Business Process Diagram Connecting Objects" on page 101.

A goal for the development of BPMN is that the notation be simple and adoptable by business analysts. Also, there is a potentially conflicting requirement that BPMN provide the power to depict complex business processes and map to BPM execution languages. To help understand how BPMN can manage both requirements, the list of BPMN graphic elements is presented in two groups.

First, there is the list of core elements that will support the requirement of a simple notation. These are the elements that define the basic look-and-feel of BPMN. Most business processes will be modeled adequately with these elements. Second, there is the entire list of elements, including the core elements, which will help support requirement of a powerful notation to handle more advanced modeling situations. And further, the graphical elements of the notation will be supported by non-graphical attributes that will provide the remaining information necessary to map to an execution language or other business modeling purposes.

8.1   BPD Core Element Set

It should be emphasized that one of the drivers for the development of BPMN is to create a simple mechanism for creating business process models, while at the same time being able to handle the complexity inherent to business processes. The approach taken to handle these two conflicting requirements was to organize the graphical aspects of the notation into specific categories. This provides a small set of notation categories so that the reader of a BPMN diagram can easily recognize the basic types of elements and understand the diagram. Within the basic categories of elements, additional variation and information can be added to support the requirements for complexity without dramatically changing the basic look and feel of the diagram. The four basic categories of elements are:

- Flow Objects
- Connecting Objects
- Swimlanes
- Artifacts

Flow Objects are the main graphical elements to define the behavior of a Business Process. There are three Flow Objects:

- Events
- Activities
- Gateways

There are three ways of connecting the Flow Objects to each other or other information. There are three Connecting Objects:

- Sequence Flow
- Message Flow
- Association

There are two ways of grouping the primary modeling elements through "Swimlanes:"

- Pools

- Lanes

Artifacts are used to provide additional information about the Process. There are three standardized Artifacts, but modelers or modeling tools are free to add as many Artifacts as required. There may be addition BPMN efforts to standardize a larger set of Artifacts for general use or for vertical markets. The current set of Artifacts include:

- Data Object

- Group

- Annotation

Table 8.2 displays a list of the core modeling elements that are depicted by the notation.

Table 8.1 - Core Modeling Elements
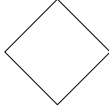
Element	Description	Notation
Event	An event is something that "happens" during the course of a business process ("Events" on page 36). These events affect the flow of the process and usually have a cause (trigger) or an impact (result). Events are circles with open centers to allow internal markers to differentiate different triggers or results. There are three types of Events, based on when they affect the flow: Start, Intermediate, and End.	
Activity	An activity is a generic term for work that company performs ("Activities" on page 53). An activity can be atomic or non-atomic (compound). The types of activities that are a part of a Process Model are: Process, Sub-Process, and Task. Tasks and Sub-Processes are rounded rectangles. Processes are contained within a Pool.	
Gateway	A Gateway is used to control the divergence and convergence of Sequence Flow ("Gateways" on page 71). Thus, it will determine branching, forking, merging, and joining of paths. Internal Markers will indicate the type of behavior control.	

Table 8.2 - BPD Core Element Set
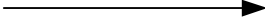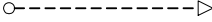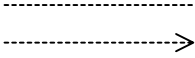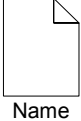
Element	Description	Notation
Sequence Flow	A Sequence Flow is used to show the order that activities will be performed in a Process ("Sequence Flow" on page 101).	(solid arrow)
Message Flow	A Message Flow is used to show the flow of messages between two participants that are prepared to send and receive them ("Message Flow" on page 103). In BPMN, two separate Pools in a Diagram will represent the two participants (e.g., business entities or business roles).	(dashed line with open circle and open arrowhead)
Association	An Association is used to associate information with Flow Objects. Text and graphical non-Flow Objects can be associated with Flow Objects. An arrowhead on the Association indicates a direction of flow (e.g., data), when appropriate ("Association" on page 105)	(dotted line and dotted line with arrowhead)
Pool	A Pool represents a Participant in a Process ("Pool" on page 88). It also acts as a "swimlane" and a graphical container for partitioning a set of activities from other Pools, usually in the context of B2B situations.	(Pool box with "Name")
Lane	A Lane is a sub-partition within a Pool and will extend the entire length of the Pool, either vertically or horizontally ("Lane" on page 91). Lanes are used to organize and categorize activities.	(Pool box with two "Name" lanes)
Data Object	Data Objects are considered Artifacts because they do not have any direct effect on the Sequence Flow or Message Flow of the Process, but they do provide information about what activities require to be performed and/or what they produce ("Data Object" on page 95).	(document icon with "Name")
Group (a box around a group of objects within the same category)	A grouping of activities that are within the same category ("Group" on page 97). This type of grouping does not affect the Sequence Flow of the activities within the group. The category name appears on the diagram as the group label. Categories can be used for documentation or analysis purposes. Groups are one way in which categories of objects can be visually displayed on the diagram.	(dashed rounded rectangle)

Table 8.2 - BPD Core Element Set

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 ("Text Annotation" on page 97).	Descriptive Text Here

8.2   BPD Extended Set

Table 8.3 displays a more extensive list of the business process concepts that could be depicted through a business process modeling notation.

Table 8.3 - BPD Extended Element Set

Element	Description	Notation
Event	An event is something that "happens" during the course of a business process ("" on page 35). 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.	Name or Source
Flow Dimension (e.g., Start, Intermediate, End) Start (None, Message, Timer, Conditional, Signal, Multiple) Intermediate (None, Message, Timer, Error, Cancel, Compensation, Conditional, Link, Signal, Multiple) End (None, Message, Error, Cancel, Compensation, Signal, Terminate, Multiple)	As the name implies, the Start Event indicates where a particular process will start ("Start" on page 36) . Intermediate Events occur between a Start Event and an End Event ("Intermediate" on page 44). They will affect the flow of the process, but will not start or (directly) terminate the process. As the name implies, the End Event indicates where a process will end ("End" on page 40).	Start Intermediate End

Table 8.3 - BPD Extended Element Set
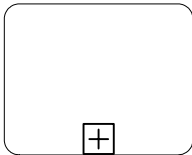
Type Dimension (e.g., None, Message, Timer, Error, Cancel, Compensation, Conditional, Link, Signal, Multiple, Terminate.)	Start and most Intermediate Events have "Triggers" that define the cause for the event ("Start" on page 36 and "Intermediate" on page 44). There are multiple ways that these events can be triggered. End Events may define a "Result" that is a consequence of a Sequence Flow ending ("End" on page 40). Start Events can only react to ("catch") a Trigger. End Events can only create ("throw") a Result. Intermediate Events can catch or throw Triggers. For the Events, Triggers that catch, the markers are unfilled, and for Triggers and Results that throw, the markers are filled.	"Catching" "Throwing" Message ⊠ ⊠ ⊠ ⊠ Timer ⏱ ⏱ Error Ⓝ Ⓝ Cancel ⊗ ⊗ Compensation ⊲⊲ ◀◀ ◀◀ Conditional ▤ ▤ Link ⇨ ⇨ Signal △ △ ▲ ▲ Terminate ● Multiple ⬠ ⬠ ⬟ ⬟
Task (Atomic)	A Task is an atomic activity that is included within a Process ("Task" on page 65). A Task is used when the work in the Process is not broken down to a finer level of Process Model detail.	
Process/Sub-Process (non-atomic)	A Sub-Process is a compound activity that is included within a Process ("Sub-Process" on page 56). 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
Collapsed Sub-Process	The details of the Sub-Process are not visible in the Diagram ("Sub-Process" on page 56). 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.	⊞

Table 8.3 - BPD Extended Element Set
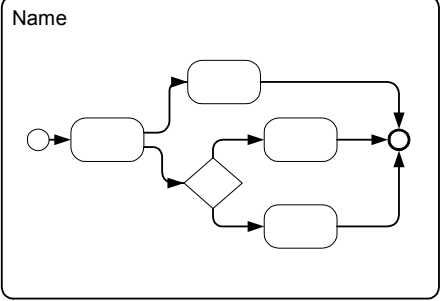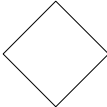
Expanded Sub-Process	The boundary of the Sub-Process is expanded and the details (a Process) are visible within its boundary ("Sub-Process" on page 56). Note that Sequence Flow cannot cross the boundary of a Sub-Process.	
Gateway	A Gateway is used to control the divergence and convergence of multiple Sequence Flow ("Gateways" on page 71). 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: • Exclusive decision and merging. Both Data-Based ("Data-Based" on page 74) and Event-Based ("Event-Based" on page 78). Data-Based can be shown with or without the "X" marker. • Inclusive decision and merging ("Inclusive Gateways" on page 81) • Complex -- complex conditions and situations (e.g., 3 out of 5; "Complex Gateways" on page 84) • Parallel forking and joining ("Parallel Gateways" on page 86) Each type of control affects both the incoming and outgoing Flow.	
Sequence Flow	A Sequence Flow is used to show the order that activities will be performed in a Process ("Sequence Flow" on page 101).	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 ("Normal Flow" on page 108).	

Table 8.3 - BPD Extended Element Set
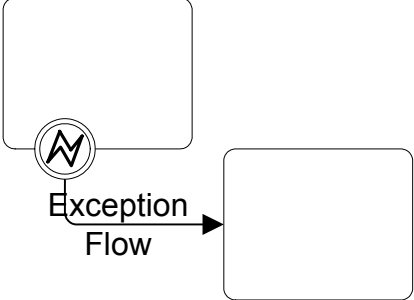
Uncontrolled flow	Uncontrolled flow refers to flow that is not affected by any conditions or does not pass through a Gateway ("Gateways" on page 71). The simplest example of this is a single Sequence Flow connecting two activities. This can also apply to multiple Sequence Flow that converge on or diverge from an activity. For each uncontrolled Sequence Flow a "Token" will flow from the source object to the target object.	
Conditional flow	Sequence Flow can have condition expressions that are evaluated at runtime to determine whether or not the flow will be used ("Sequence Flow" on page 101). 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).	
Default flow	For Data-Based Exclusive Decisions or Inclusive Decisions, one type of flow is the Default condition flow ("Sequence Flow" on page 101). 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).	
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 ("Exception Flow" on page 131).	

Table 8.3 - BPD Extended Element Set
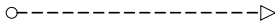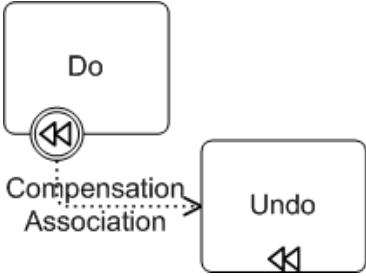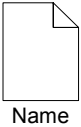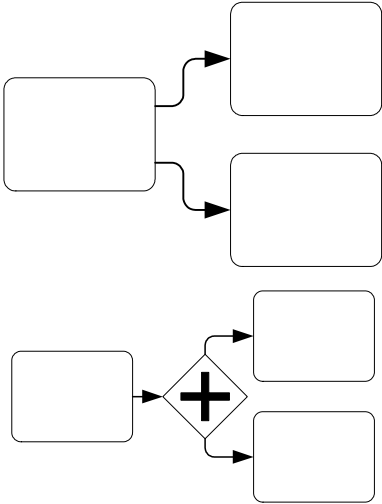
Message Flow	A Message Flow is used to show the flow of messages between two entities that are prepared to send and receive them ("Message Flow" on page 103). In BPMN, two separate Pools in the Diagram will represent the two entities.	
Compensation Association	Compensation Association occurs outside the Normal Flow of the Process and is based upon an event (a Compensation Intermediate Event) that is triggered through the failure of a Transaction or a Compensate Event ("Compensation Association" on page 133). The target of the Association must be marked as a Compensation Activity.	
Data Object	Data Objects are considered Artifacts because they do not have any direct effect on the Sequence Flow or Message Flow of the Process, but they do provide information about what activities require to be performed and/or what they produce ("Data Object" on page 95).	
Fork	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; "Forking Flow" on page 111). It is a place in the Process where activities can be performed concurrently, rather than sequentially. 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 Gateway can be used (see figure bottom-right). This will be used rarely, usually in combination with other Gateways.	

Table 8.3 - BPD Extended Element Set
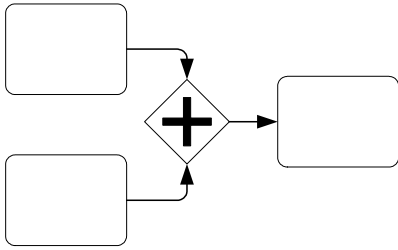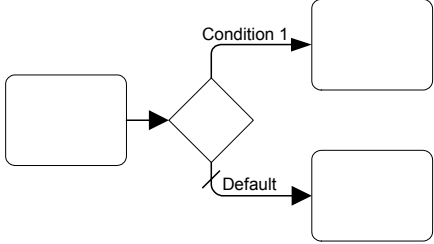
Join	BPMN uses the term "join" to refer to the combining of two or more parallel paths into one path (also known as an AND-Join or synchronization; "Joining Flow" on page 114). A Parallel Gateway is used to show the joining of multiple Flow.	
Decision, Branching Point	Decisions are Gateways within a business process where the flow of control can take one or more alternative paths ("Gates" on page 74).	See next five rows.
Exclusive	An Exclusive Gateway restricts the flow such that only one of a set of alternatives may be chosen during runtime ("Gates" on page 74). There are two types of Exclusive Gateways: Data-based and Event-based.	
Data-Based	This Decision represents a branching point where Alternatives are based on conditional expressions contained within the outgoing Sequence Flow ("Data-Based" on page 74). Only one of the Alternatives will be chosen.	

Table 8.3 - BPD Extended Element Set

Event-Based	This Decision represents a branching point where Alternatives are based on an Event that occurs at that point in the Process ("Event-Based" on page 78). 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).	
Inclusive	This Decision represents a branching point where Alternatives are based on conditional expressions contained within the outgoing Sequence Flow ("Inclusive Gateways" on page 81). 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. A Default Condition could be used to ensure 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 Inclusive Gateway (see bottom-right picture).	

Table 8.3 - BPD Extended Element Set
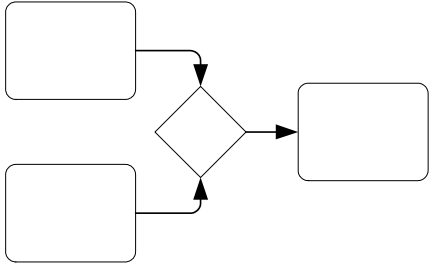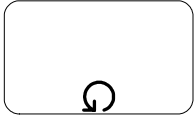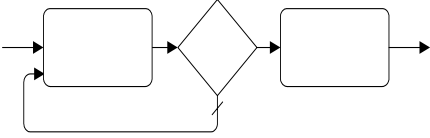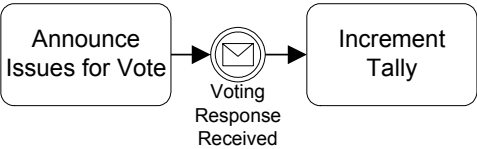
Merging	BPMN uses the term "merge" to refer to the exclusive combining of two or more paths into one path (also known as an a OR-Join; "Merging Flow" on page 118). A Merging Exclusive Gateway is used to show the merging of multiple Flow. If all the incoming flow is alternative, then a Gateway is not needed. That is, uncontrolled flow provides the same behavior.	
Looping	BPMN provides 2 (two) mechanisms for looping within a Process.	See Next Two Figures
Activity Looping	The attributes of Tasks and Sub-Processes will determine if they are repeated or performed once ("Looping" on page 122). There are two types of loops: Standard and Multi-Instance. A small looping indicator will be displayed at the bottom-center of the activity.	
Sequence Flow Looping	Loops can be created by connecting a Sequence Flow to an "upstream" object ("Looping" on page 122). An object is considered to be upstream if that object has an outgoing Sequence Flow that leads to a series of other Sequence Flow, the last of which is an incoming Sequence Flow for the original object.	
Multiple Instances	The attributes of Tasks and Sub-Processes will determine if they are repeated or performed once ("Looping" on page 122). A small parallel indicator will be displayed at the bottom-center of the activity.	
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 ("Intermediate" on page 44). 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.	

Table 8.3 - BPD Extended Element Set
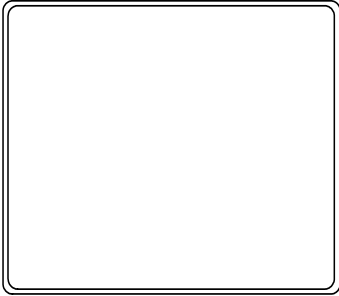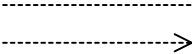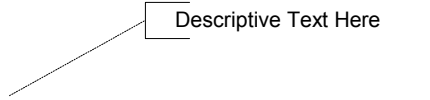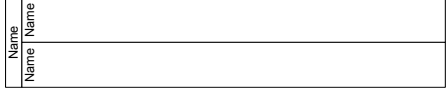
Transaction	A transaction is a Sub-Process that is supported by a special protocol that insures that all parties involved have complete agreement that the activity should be completed or cancelled ("Sub-Process Behavior as a Transaction" on page 63). The attributes of the activity will determine if the activity is a transaction. A double-lined boundary indicates that the Sub-Process is a Transaction.	
Nested/Embedded Sub-Process (Inline Block)	A nested (or embedded) Sub-Process is an activity that shares the same set of data as its parent process ("Embedded Sub-Process" on page 59). 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
Group (a box around a group of objects within the same category)	A grouping of activities that are within the same category ("Group" on page 97). This type of grouping does not affect the Sequence Flow of the activities within the group. The category name appears on the diagram as the group label. Categories can be used for documentation or analysis purposes. Groups are one way in which categories of objects can be visually displayed on the diagram.	
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 ("Sequence Flow Jumping (Off-Page Connectors and Go To Objects)" on page 125). A Link Intermediate Event can be used as an Off-Page Connector.	
Association	An Association is used to associate information with Flow Objects ("Association" on page 105). Text and graphical non-Flow Objects can be associated with the Flow Objects.	

Table 8.3 - BPD Extended Element Set

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 ("Text Annotation" on page 97).	Descriptive Text Here
Pool	A Pool represents a Participant in a Process ("Pool" on page 88). It is also acts as a "swimlane" and a graphical container for partitioning a set of activities from other Pools, usually in the context of B2B situations.	Name
Lanes	A Lane is a sub-partition within a Pool and will extend the entire length of the Pool, either vertically or horizontally ("Lane" on page 91). Lanes are used to organize and categorize activities within a Pool.	Name Name / Name Name

8.3   Use of Text, Color, Size, and Lines in a Diagram

Text Annotation objects can be used by the modeler to display additional information about a Process or attributes of the objects within the Process.

- Flow objects and Flow MAY have labels (e.g., its name and/or other attributes) placed inside the shape, or above or below the shape, in any direction or location, depending on the preference of the modeler or modeling tool vendor.

- The fills that are used to for the graphical elements MAY be white or clear.
  - The notation MAY be extended to use other fill colors to suit the purpose of the modeler or tool (e.g., to highlight the value of an object attribute).

- Flow objects and markers MAY be of any size that suits the purposes of the modeler or modeling tool.

- The lines that are used to draw the graphical elements MAY be black.
  - The notation MAY be extended to use other line colors to suit the purpose of the modeler or tool (e.g., to highlight the value of an object attribute).
  - The notation MAY be extended to use other line styles to suit the purpose of the modeler or tool (e.g., to highlight the value of an object attribute) with the condition that the line style MUST NOT conflict with any current BPMN defined line style. Thus, the line styles of Sequence Flow, Message Flow, and Associations MUST NOT be modified.

8.4   Flow Object Connection Rules

An incoming Sequence Flow can connect to any location on a Flow Object (left, right, top, or bottom). Likewise, an outgoing Sequence Flow can connect from any location on a Flow Object (left, right, top, or bottom). Message Flow also have this capability. BPMN allows this flexibility, however, we also recommend that modelers use judgment or best practices in how Flow Objects should be connected so that readers of the Diagrams will find the behavior clear and easy

to follow. This is even more important when a Diagram contains Sequence Flow and Message Flow. In these situations it is best to pick a direction of Sequence Flow, either left to right or top to bottom, and then direct the Message Flow at a 90° angle to the Sequence Flow. The resulting Diagrams will be much easier to understand.

8.4.1 Sequence Flow Rules

Table 8.4 displays the BPMN Flow Objects and shows how these objects can connect to one another through Sequence Flow. The ⬈ symbol indicates that the object listed in the row can connect to the object listed in the column. The quantity of connections into and out of an object is subject to various configuration dependencies are not specified here. Refer to the sections in the next chapter for each individual object for more detailed information on the appropriate connection rules. *Note that if a sub-process has been expanded within a Diagram, the objects within the sub-process cannot be connected to objects outside of the sub-process. Nor can Sequence Flow cross a Pool boundary.*

Table 8.4 - Sequence Flow Connection Rules

From\To	◯	Name +	Name	◇	◎	⬤
◯		⬈	⬈	⬈	⬈	⬈
Name +		⬈	⬈	⬈	⬈	⬈
Name		⬈	⬈	⬈	⬈	⬈
◇		⬈	⬈	⬈	⬈	⬈
◎		⬈	⬈	⬈	⬈	⬈
⬤						
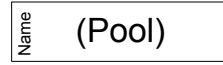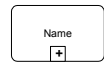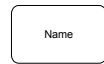
Note – Only those objects that can have incoming and/or outgoing Sequence Flow are shown in the table. Thus, Pool, Lane, Data Object, and Text Annotation are not listed in the table.

8.4.2 Message Flow Rules

Table 8.5 displays the BPMN modeling objects and shows how these objects can connect to one another through Message Flow. The ⬈ symbol indicates that the object listed in the row can connect to the object listed in the column. The quantity of connections into and out of an object is subject to various configuration dependencies are not specified here. Refer to the sections in the next chapter for each individual object for more detailed information on the appropriate connection rules. *Note that Message Flow cannot connect to objects that are within the same Pool.*

Table 8.5 - Message Flow Connection Rules

From\To	✉	Name (Pool)	Name ⊞	Name	◉✉	⬤✉
✉						
Name (Pool)	↗	↗	↗	↗	↗	
Name ⊞	↗	↗	↗	↗	↗	
Name	↗	↗	↗	↗	↗	
◉✉	↗	↗	↗	↗	↗	
⬤✉	↗	↗	↗	↗	↗	

Note – Only those objects that can have incoming and/or outgoing Message Flow are shown in the table. Thus, Lane, Gateway, Data Object, and Text Annotation are not listed in the table.

8.5   Business Process Diagram Attributes

The following table displays the set of attributes of a Business Process Diagram:

Table 8.6 - Business Process Diagram Attributes

Attributes	Description
Id: Object	This is a unique Id that distinguishes the Diagram from other Diagrams.
Name: String	Name is an attribute that is text description of the Diagram.
Version (0-1) : String	This defines the Version number of the Diagram.
Author (0-1) : String	This holds the name of the author of the Diagram.
Language (0-1) : String	This holds the name of the language in which text is written. The default is English.
QueryLanguage (0-1) : String	A Language MAY be provided so that the syntax of queries used in the Diagram can be understood.
CreationDate (0-1) : Date	This defines the date on which the Diagram was created (for the current Version).
ModificationDate (0-1) : Date	This defines the date on which the Diagram was last modified (for this Version).

Table 8.6 - Business Process Diagram Attributes

Attributes	Description
Pools (1-n) : Pool	A BPD SHALL contain one or more Pools. The boundary of one of the Pools MAY be invisible (especially if there is only one Pool in the Diagram). Refer to the Section B.8.2, "Pool," on page 268 for more information about Pools.
Documentation (0-1) : String	The modeler MAY add optional text documentation about the Diagram.

8.6   Processes

A **Process** is an activity performed within or across companies or organizations. In BPMN a Process is depicted as a graph of Flow Objects, which are a set of other activities and the controls that sequence them. The concept of process is intrinsically hierarchical. Processes may be defined at any level from enterprise-wide processes to processes performed by a single person. Low-level processes may be grouped together to achieve a common business goal.

Note that BPMN defines the term Process fairly specifically and defines a Business Process more generically as a set of activities that are performed within an organization or across organizations. Thus a Business Process, as shown in a Business Process Diagram, may contain more than one separate Process. Each Process may have its own Sub-Processes and would be contained within a Pool (Section B.8.2, "Pool," on page 268). The individual Processes would be independent in terms of Sequence Flow, but could have Message Flow connecting them.

8.6.1   Attributes

The following table displays the set of attributes of a Process, and which extends the set of common BPMN Element attributes (see Table B.49):

Table 8.7 - Process Attributes

Attributes	Description
Name : String	Name is an attribute that is a text description of the object.
ProcessType (None | Private | Abstract | Collaboration) None : String	ProcessType is an attribute that provides information about which lower-level language the Pool will be mapped. By default, the ProcessType is None (or undefined).
Status (None | Ready | Active | Cancelled | Aborting | Aborted | Completing | Completed) None : String	The Status of a Process is determined when the Process is being executed by a process engine. The Status of a Process can be used within Assignment Expressions.
GraphicalElements (0-n) : Object	The GraphicalElements attribute identifies all of the objects (e.g., Events, Activities, Gateways, and Artifacts) that are contained within the Business Process.
Performers (0-n) : String	One or more Performers MAY be entered. The Performers attribute defines the resource that will be responsible for the Process. The Performers entry could be in the form of a specific individual, a group, an organization role or position, or an organization.

Table 8.7 - Process Attributes

Attributes	Description
Assignments (0-n) **:** Assignment	One or more assignment expressions MAY be made for the object. The Assignment SHALL be performed as defined by the AssignTime attribute. The details of the Assignment is defined in the Section B.11.3, "Assignment," on page 277.
Properties (0-n) : Property	Modeler-defined Properties MAY be added to a Process. These Properties are "local" to the Process. All Tasks, Sub-Process objects, and Sub-Processes that are embedded SHALL have access to these Properties. The fully delineated name of these properties are "<process name>.<property name>" (e.g., "Add Customer.Customer Name"). If a process is embedded within another Process, then the fully delineated name SHALL also be preceded by the Parent Process name for as many Parents there are until the top level Process. Further details about the definition of a Property can be found in the Section B.11.15, "Property," on page 285.
InputSets (0-n) : InputSet	The InputSets attribute defines the data requirements for input to the Process. Zero or more InputSets MAY be defined. Each Input set is sufficient to allow the Process to be performed (if it has first been instantiated by the appropriate signal arriving from an incoming Sequence Flow). Further details about the definition of an Input-Set can be found in Section B.11.10, "InputSet," on page 282.
OutputSets (0-n) : OutputSet	The OutputSets attribute defines the data requirements for output from the Process. Zero or more OutputSets MAY be defined. At the completion of the Process, only one of the OutputSets may be produced--It is up to the implementation of the Process to determine which set will be produced. However, the IORules attribute MAY indicate a relationship between an OutputSet and an InputSet that started the Process. Further details about the definition of an OutputSet can be found in Section B.11.13, "OutputSet," on page 284.
AdHoc False : Boolean	AdHoc is a boolean attribute, which has a default of False. This specifies whether the Process is Ad Hoc or not. The activities within an Ad Hoc Process are not controlled or sequenced in a particular order, their performance is determined by the performers of the activities. If set to True, then the Ad Hoc marker SHALL be placed at the bottom center of the Process or the Sub-Process shape for Ad Hoc Processes.
[AdHoc = True only] **AdHocOrdering** (0-1) (Sequential | Parallel) Parallel : String	If the Process is Ad Hoc (the AdHoc attribute is True), then the AdHocOrdering attribute MUST be included. This attribute defines if the activities within the Process can be performed in Parallel or must be performed sequentially. The default setting is Parallel and the setting of Sequential is a restriction on the performance that may be required due to shared resources.
[AdHoc = True only] **AdHocCompletionCondition** (0-1) : Expression	If the Process is Ad Hoc (the AdHoc attribute is True), then the AdHocCompletionCondition attribute MUST be included. This attribute defines the conditions when the Process will end.

9   Business Process Diagram Graphical Objects

This section details the graphical representation and the semantics of the behavior of Business Process Diagram graphical elements. "Mapping BPMN Elements and Diagrams to BPEL4WS" on page 145 for more information about how these elements map to execution languages.

9.1   Common BPMN Element Attributes

The following table displays a set of common attributes for BPMN elements (graphical elements and supporting elements):

Table 9.1 - Common BPMN Element Attributes

Attributes	Description
Id : Object	This is a unique Id that identifies the object from other objects within the Diagram.
Categories (0-n) : Category	The modeler MAY add one or more defined Categories, which have user-defined semantics, and that can be used for purposes such as reporting and analysis. The details of Catogories is defined in "Category" on page 277.
Documentation (0-1) : String	The modeler MAY add text documentation about the object.

These attributes are used for Graphical Elements [Flow Objects (Section 9.2, "Common Flow Object Attributes," on page 35), Connecting Objects (Section 10.1, "Graphical Connecting Objects," on page 101), Swimlanes (Section 9.6, "Swimlanes (Pools and Lanes)," on page 87), and Artifacts (Section 9.7, "Artifacts," on page 94)], and Supporting Elements (Section B.11, "Supporting Elements," on page 274).

9.2   Common Flow Object Attributes

The following table displays a set of common attributes for BPMN Flow Objects (Events, Activities, and Gateways), and which extends the set of common BPMN Element attributes (see Table 9.1):

Table 9.2 - Common Flow Object Attributes

Attributes	Description
Name : String	Name is an attribute that is text description of the object.
Assignments (0-n) : Assignment	One or more assignment expressions MAY be made for the object. For activities (Task, Sub-Process, and Process), the Assignment SHALL be performed as defined by the AssignTime attribute. The Details of the Assignment is defined in Section B.11.3, "Assignment," on page 277.

9.3   Events

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 or an impact. The term "event" is general enough to cover many things in a business process. The start of an activity, the end of an activity, the change of state of a document, a message that arrives, etc., all could be considered events. However, BPMN has restricted the use of events to include only those types of events that will affect the sequence or timing of activities of a process. BPMN further categorizes Events into three main types: Start, Intermediate, and End.

Start and most Intermediate Events have "Triggers" that define the cause for the event. There are multiple ways that these events can be triggered ("Start Event Triggers" on page 38 and "Intermediate Event Triggers" on page 45). End Events may define a "Result" that is a consequence of a Sequence Flow ending. There are multiple types of Results that can be defined ("End Event Results" on page 41).

All Events share the same shape footprint, a small circle. Different line styles, as shown below, distinguish the three types of flow Events. All Events also have an open center so that BPMN-defined and modeler-defined icons can be included within the shape to help identify the Trigger or Result of the Event.

9.3.1 Common Event Attributes

The following table displays the set of attributes common to the three types of Events, and which extends the set of common Flow Object attributes (see Table 9.2):

Table 9.3 - Common Event Attributes

Attributes	Description
EventType (Start | End | Intermediate) Start : String	The EventType MUST be of type Start, End, or Intermediate.

9.3.2 Start

As the name implies, the Start Event indicates where a particular Process will start. In terms of Sequence Flow, the Start Event starts the flow of the Process, and thus, will not have any incoming Sequence Flow—no Sequence Flow can connect to a Start Event.

The Start Event shares the same basic shape of the Intermediate Event and End Event, a circle with an open center so that markers can be placed within the circle to indicate variations of the Event.

◆ A Start Event is a circle that MUST be drawn with a single thin line (see Figure 9.1).

◆ The use of text, color, size, and lines for a Start Event MUST follow the rules defined in Section 8.3, "Use of Text, Color, Size, and Lines in a Diagram," on page 29 with the exception that:

◆ The thickness of the line MUST remain thin so that the Start Event may be distinguished from the Intermediate and End Events.

Figure 9.1 - A Start Event

Throughout this document, we will discuss how Sequence Flow proceeds within a Process. To facilitate this discussion, we will employ the concept of a "**Token**" that will traverse the Sequence Flow and pass through the Flow Objects in the Process. The behavior of the Process can be described by tracking the path(s) of the Token through the Process. A Token will have a unique identity, called a TokenId set, that can be used to distinguish multiple Tokens that may exist because of concurrent Process instances or the dividing of the Token for parallel processing within a single Process instance. The parallel dividing of a Token creates a lower level of the TokenId set. The set of all levels of TokenId will identify a Token.

A Start Event generates a Token that must eventually be consumed at an End Event (which may be implicit if not graphically displayed). The path of Tokens should be traceable through the network of Sequence Flow, Gateways, and activities within a Process. There MUST NOT be any implicit flow during the course of <u>normal</u> Sequence Flow (i.e., there should always be either Sequence Flow or a graphical indicator, such as an Intermediate Event to show all the potential paths of Tokens). An example of implicit flow is when a Token arrives at a Gateway, but none of the Gates are valid, the Token would then (implicitly) pass to the end of the Process, which occurs with some modeling notations.Tokens can also be directed through exception handling Intermediate Events, which act like a forced end to an activity. Note: A Token does not traverse the Message Flow since it is a Message that is passed down those Flow (as the name implies).

Semantics of the Start Event include:

♦   A Start Event is OPTIONAL: a Process level—a top-level Process or an expanded Sub-Process—MAY (is no required to) have a Start Event:

Note – A BPD may have more than one Process level (i.e., it can include Expanded Sub-Processes). The use of Start and End Events is independent for each level of the Diagram.

♦   If a Process is complex and/or the starting conditions are not obvious, then it is RECOMMENDED that a Start Event be used

♦   If a Start Event is not used, then the implicit Start Event for the Process SHALL NOT have a Trigger.

♦   If there is an End Event, then there MUST be at least one Start Event.

♦   If the Start Event is used, then there MUST NOT be other flow elements that do not have incoming Sequence Flow—all other Flow Objects MUST be a target of at least one Sequence Flow.

♦   Exceptions to this are activities that are defined as being Compensation activities (have the Compensation Marker). Compensation activities MUST NOT have any incoming Sequence Flow, even if there is a Start Event in the Process level. See Section 10.3, "Compensation Association," on page 133 for more information on Compensation activities.

♦   An exception to this is the Intermediate Event, which MAY be without an incoming Sequence Flow (when attached to an activity boundary).

♦   If the Start Event is *not* used, then all Flow Objects that do not have an incoming Sequence Flow (i.e., are not a target of a Sequence Flow) SHALL be instantiated when the Process is instantiated. There is an assumption that there is only one implicit Start Event, meaning that all the starting Flow Objects will start at the same time.

♦   Exceptions to this are activities that are defined as being Compensation activities (have the Compensation Marker). Compensation Activities are not considered a part of the Normal Flow and MUST NOT be

instantiated when the Process is instantiated.

- ◆ There MAY be multiple Start Events for a given Process level.
  - ◆ Each Start Event is an independent event. That is, a Process Instance SHALL be generated when the Start Event is triggered.
    - ◆ If the Process is used as a Sub-Process and there are multiple None Start Events, then when flow is transferred from the parent Process to the Sub-Process, only one of the Sub-Process's Start Events will be Triggered. The TargetRef attribute of the Sequence Flow incoming to the Sub-Process object can be extended to identify the appropriate Start Event (as defined in the Sub-Process's "Sequence Flow Connections" on page 64).

Note – The behavior of Process may be harder to understand if there are multiple Start Events. It is RECOMMENDED that this feature be used sparingly and that the modeler be aware that other readers of the Diagram may have difficulty understanding the intent of the Diagram.

When the trigger for a Start Event occurs, a new Process will be instantiated and a Token will be generated for each outgoing Sequence Flow from that event. The TokenId set for each of the Tokens will be established such that it can be identified that the Tokens are all from the same parallel Fork and the number of Tokens in the group. These Tokens will begin their flow and not wait for any other Start Event to be triggered.

If there is a dependency for more than one Event to happen before a Process can start (e.g., two messages are required to start), then the Start Events must flow to the same activity within that Process. The attributes of the activity would specify when the activity could begin. If the attributes specify that the activity must wait for all inputs, then all Start Events will have to be triggered before the Process begins ("Attributes" on page 39 (for sub-processes) and "Attributes" on page 66 (for Tasks) for more information about activity attributes). In addition, a correlation mechanism will be required so that different triggered Start Events will apply to the same process instance.

Start Event Triggers

There are many ways that business process can be started (instantiated). The Trigger for a Start Event is designed to show the general mechanism that will instantiate that particular Process. There are six (6) types of Start Events in BPMN: None, Message, Timer, Conditional, Signal, and Multiple.

Table 9.4 displays the types of Triggers and the graphical marker that will be used for each:

Table 9.4 - Start Event Types

Trigger	Description	Marker
None	The modeler does not display the type of Event. It is also used for a Sub-Process that starts when the flow is triggered by its Parent Process.	○
Message	A Message arrives from a participant and triggers the start of the Process.	✉
Timer	A specific time-date or a specific cycle (e.g., every Monday at 9am) can be set that will trigger the start of the Process.	🕐
Conditional	This type of event is triggered when a Condition such as "S&P 500 changes by more than 10% since opening," or "Temperature above 300C" become true. The ConditionExpression for the Event must become false and then true before the Event can be triggered again.	☰
Signal	A signal arrives that has been broadcast from another Process and triggers the start of the Process. Note that the Signal is not a Message, which has a specific target for the Message. Multiple Processes can have Start Events that are triggered from the same broadcasted Signal. The attributes of a Signal can be found in Section B.11.17, "Signal," on page 285.	△
Multiple	This means that there are multiple ways of triggering the Process. Only one of them will be required to start the Process. The attributes of the Start Event will define which of the other types of Triggers apply.	⬠

Attributes

The following table displays the set of attributes of a Start Event, which extends the set of common Event attributes (see Table 9.5):

Table 9.5 - Start Event Attributes

Attributes	Description
Trigger (0-n) : EventDetail	Trigger (EventDetail) is an attribute that defines the type of trigger expected for a Start Event. Of the set of EventDetailTypes (see Section 9.3.5, "Event Details," on page 49), only four (4) can be applied to a Start Event: Message, Timer, Conditional, and Signal (see Table 9.4). If there is no EventDetail is defined, then this is considered a None End Event and the Event will not have an internal marker (see Table 9.4). If there is more than one EventDetail is defined, this is considered a Multiple End Event and the Event will have the star internal marker (see Table 9.4).

Sequence Flow Connections

See Section 8.4.1, "Sequence Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Sequence Flow.

◆ A Start Event MUST NOT be a target for Sequence Flow; it MUST NOT have incoming Sequence Flow.

   ◆ An exception to this is when a Start Event is used in an Expanded Sub-Process and is attached to the boundary of that Sub-Process. In this case, a Sequence Flow from the higher-level Process MAY connect to that Start Event in lieu of connecting to the actual boundary of the Sub-Process (see Figure 10.15).

◆ A Start Event MUST be a source for Sequence Flow.

◆ Multiple Sequence Flow MAY originate from a Start Event. For each Sequence Flow that has the Start Event as a source, a new parallel path SHALL be generated.

   ◆ The Condition attribute for all outgoing Sequence Flow MUST be set to None.

   ◆ When a Start Event is not used, then all Flow Objects that do not have an incoming Sequence Flow SHALL be the start of a separate parallel path.

Each path will have a separate unique Token that will traverse the Sequence Flow.

Message Flow Connections

See Section 8.4.2, "Message Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Message Flow.

Note – All Message Flow must connect two separate Pools. They can connect to the Pool boundary or to Flow Objects within the Pool boundary. They cannot connect two objects within the same Pool.

◆ A Start Event MAY be the target for Message Flow; it can have 0 (zero) or more incoming Message Flow. Each Message Flow arriving at a Start Event represents an instantiation mechanism (a Trigger) for the process. Only one of the Triggers is required to start a new Process.

   ◆ The Trigger attribute of the Start Event MUST be set to "Message" or "Multiple" if there are any incoming Message Flow.

      ◆ The Trigger attribute of the Start Event MUST be set to "Multiple" if there are more than one incoming Message Flow.

◆ A Start Event MUST NOT be a source for Message Flow; it MUST NOT have outgoing Message Flow.

9.3.3 End

As the name implies, the End Event indicates where a process will end. In terms of Sequence Flow, the End Event ends the flow of the Process, and thus, will not have any outgoing Sequence Flow—no Sequence Flow can connect from an End Event.

The End Event shares the same basic shape of the Start Event and Intermediate Event, a circle with an open center so that markers can be placed within the circle to indicate variations of the Event.

◆ An End Event is a circle that MUST be drawn with a single thick black line (see Figure 9.2).

   ◆ The use of text, color, size, and lines for an End Event MUST follow the rules defined in Section 8.3, "Use of Text, Color, Size, and Lines in a Diagram," on page 29 with the exception that:

      ◆ The thickness of the line MUST remain thick so that the End Event may be distinguished from the Intermediate and Start Events.

Figure 9.2 - End Event

To continue discussing how flow proceeds throughout the process, an End Event consumes a Token that had been generated from a Start Event within the same level of Process. If parallel Sequence Flow target the End Event, then the Tokens will be consumed as they arrive. All the Tokens that were generated within the Process must be consumed by an End Event before the Process has been completed. In other circumstances, if the Process is a Sub-Process, it can be stopped prior to normal completion through interrupting Intermediate Events (See Section 10.2.2, "Exception Flow," on page 131 for more details). In this situation the Tokens will be consumed by an Intermediate Event attached to the boundary of the Sub-Process.

Semantics of the End Event include:

- There MAY be multiple End Events within a single level of a process.

- An End Event is OPTIONAL: a given Process level—a top-level Process or an expanded Sub-Process—MAY (is not required to) have this shape:

  - If an End Event is not used, then the implicit End Event for the Process SHALL NOT have a Result.

  - If there is a Start Event, then there MUST be at least one End Event.

  - If an End Event is used, then there MUST NOT be other flow elements that do not have any outgoing Sequence Flow—all other Flow Objects MUST be a source of at least one Sequence Flow.

    - Exceptions to this are activities that are defined as being Compensation activities (have the Compensation Marker). Compensation Activities MUST NOT have any outgoing Sequence Flow, even if there is an End Event in the Process level. Section 10.3, "Compensation Association," on page 133 for more information on Compensation activities.

  - If the End Event is not used, then all Flow Objects that do not have any outgoing Sequence Flow (i.e., are not a source of a Sequence Flow) mark the end of a path in the Process. However, the process MUST NOT end until all parallel paths have completed.

    - Exceptions to this are activities that are defined as being Compensation activities (have the Compensation Marker). Compensation Activities are not considered a part of the Normal Flow and MUST NOT mark the end of the Process.

Note – A BPD may have more than one Process level (i.e., it can include Expanded Sub-Processes). The use of Start and End Events is independent for each level of the Diagram.

For Processes without an End Event, a Token entering a path-ending Flow Object will be consumed when the processing performed by the object is completed (i.e., when the path has completed), as if the Token had then gone on to reach an End Event. When all Tokens for a given instance of the Process are consumed, then the Process will reach a state of being completed.

End Event Results

There are eight (8) types of End Events in BPMN: None, Message, Error, Cancel, Compensation, Signal, Terminate, and Multiple. These types define the consequence of reaching an End Event. This will be referred to as the End Event Result.

Table 9.6 displays the types of Results and the graphical marker that will be used for each:

Table 9.6 - End Event Types

Result	Description	Marker
None	The modeler does not display the type of Event. It is also used to show the end of a Sub-Process that ends, which causes the flow goes back to its Parent Process.	
Message	This type of End indicates that a message is sent to a participant at the conclusion of the Process.	
Error	This type of End indicates that a named Error should be generated. The Error will be caught by the Error intermediate event with the same ErrorCode or no ErrorCode which is on the boundary of the nearest enclosing parent activity (hierarchically). The behavior of the process is unspecified if no activity in the has such an Error intermediate event. The system executing the process may define additional Error handling in this case, a common one being termination of the process instance.	
Cancel	This type of End is used within a Transaction Sub-Process. It will indicate that the Transaction should be cancelled and will trigger a Cancel Intermediate Event attached to the Sub-Process boundary. In addition, it will indicate that a Transaction Protocol Cancel message should be sent to any Entities involved in the Transaction.	
Compensation	This type of End indicates that a Compensation is necessary. If an activity is identified, then that is the activity that will be compensated. Otherwise, all activities that have completed within the Process, starting with the top-level Process and including all Sub-Processes, are subject to compensation, proceeding in reverse order. To be compensated, an activity MUST have a Compensation Intermediate Event attached to its boundary.	
Signal	This type of End indicates that a Signal will be broadcasted when the End has been reached. Note that the Signal, which is broadcast to any Process that can receive the Signal, can be sent across Process levels or Pools, but is not a Message (which has a specific Source and Target). The attributes of a Signal can be found in Section B.11.17, "Signal," on page 285.	
Terminate	This type of End indicates that all activities in the Process should be immediately ended. This includes all instances of Multi-Instances. The Process is ended without compensation or event handling.	
Multiple	This means that there are multiple consequences of ending the Process. All of them will occur (e.g., there might be multiple messages sent). The attributes of the End Event will define which of the other types of Results apply.	

Attributes

The following table displays the set of attributes of a End Event, which extends the set of common Event attributes (see Table 9.3):

Table 9.7 - End Event Attributes

Attributes	Description
Result (0-n) : EventDetail	Result (EventDetail) is an attribute that defines the type of result expected for an End Event. Of the set of EventDetailTypes (see Section 9.3.5, "Event Details," on page 49), only six (6) can be applied to an End Event: Message, Error, Cancel, Compensation, Signal, and Terminate (see Table 9.6). If there is no EventDetail is defined, then this is considered a None End Event and the Event will not have an internal marker (see Table 9.6). If there is more than one EventDetail is defined, this is considered a Multiple End Event and the Event will have the star internal marker (see Table 9.6).

Sequence Flow Connections

Section 8.4.1, "Sequence Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Sequence Flow.

◆ An End Event MUST be a target for Sequence Flow.

◆ An End Event MAY have multiple incoming Sequence Flow.

The Flow MAY come from either alternative or parallel paths. For modeling convenience, each path MAY connect to a separate End Event object. The End Event is used as a Sink for all Tokens that arrive at the Event. All Tokens that are generated at the Start Event for that Process must eventually arrive at an End Event. The Process will be in a *running* state until all Tokens are consumed.

◆ An End Event MUST NOT be a source for Sequence Flow; that is, there MUST NOT be outgoing Sequence Flow.

  ◆ An exception to this is when an End Event is used in an Expanded Sub-Process and is attached to the boundary of that Sub-Process. In this case, a Sequence Flow from the higher-level Process MAY connect from that End Event in lieu of connecting from the actual boundary of the Sub-Process (see Figure 10.15).

Message Flow Connections

See Section 8.4.2, "Message Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Message Flow.

Note – All Message Flow must connect two separate Pools. They can connect to the Pool boundary or to Flow Objects within the Pool boundary. They cannot connect two objects within the same Pool.

◆ An End Event MUST NOT be the target for Message Flow; it can have no incoming Message Flow. If the Intermediate Event has an incoming Message Flow, then it MUST NOT have an outgoing Message Flow.

◆ An Intermediate Event of type Message, if it is used within Normal Flow, MAY be the source for Message Flow; it can have one outgoing Message Flow. If the Intermediate Event has an outgoing Message Flow, then it MUST NOT have an incoming Message Flow.

9.3.4 Intermediate

As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the Start and End of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. Intermediate Events can be used to:

- Show where messages are expected or sent within the Process,

- Show delays are expected within the Process,

- Disrupt the Normal Flow through exception handling, or

- Show the extra work required for compensation.

The Intermediate Event shares the same basic shape of the Start Event and End Event, a circle with an open center so that markers can be placed within the circle to indicate variations of the Event.

- ◆ An Intermediate Event is a circle that MUST be drawn with a double thin black line. (see Figure 9.3).

    - ◆ The use of text, color, size, and lines for an Intermediate Event MUST follow the rules defined in Section 8.3, "Use of Text, Color, Size, and Lines in a Diagram," on page 29 with the exception that:

        - ◆ The thickness of the line MUST remain double so that the Intermediate Event may be distinguished from the Start and End Events.

Figure 9.3 - **Intermediate Event**

One use of Intermediate Events is to represent exception or compensation handling. This will be shown by placing the Intermediate Event on the boundary of a Task or Sub-Process (either collapsed or expanded). Figure 9.4 displays an example of an Intermediate Event attached to a Task. The Intermediate Event can be attached to any location of the activity boundary and the outgoing Sequence Flow can flow in any direction. However, in the interest of clarity of the Diagram, we recommend that the modeler choose a consistent location on the boundary. For example, if the Diagram orientation is horizontal, then the Intermediate Events can be attached to the bottom of the activity and the Sequence Flow directed down, then to the right. If the Diagram orientation is vertical, then the Intermediate Events can be attached to the left or right side of the activity and the Sequence Flow directed to the left or right, then down.
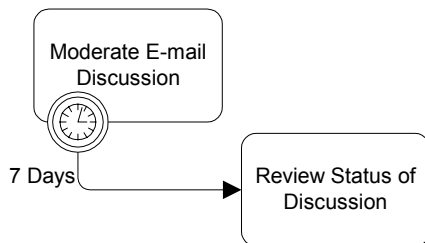
Figure 9.4 - Task with an Intermediate Event attached to its boundary

Intermediate Event Triggers

There are nine (9) types of Intermediate Events in BPMN: None, Message, Timer, Error, Cancel, Compensation, Conditional, Link, Signal, and Multiple. Each type of Intermediate Event will have a different icon placed in the center of the Intermediate Event shape to distinguish one from another.

An Intermediate Event that is placed within the normal flow of a Process can be used for one of two purposes. The Event can respond to ("catch") the Event Trigger or the Event can be used to set off ("throw") the Event Trigger. An Intermediate Event that is attached to the boundary of an Activity can only be used to "catch" the Event Trigger.

When a Token arrives at an Intermediate Event that is placed within the normal flow of a Process, one of two things will happen. If the Event is used to "throw" the Event Trigger, then Trigger of the Event will immediately occur (e.g., the Message will be sent) and the Token will move down the outgoing Sequence Flow. If the Event is used to "catch" the Event Trigger, then the Token will remain at the Event until the Trigger occurs (e.g., the Message is received). Then the Token will move down the outgoing Sequence Flow.

Table 9.8 displays the types of Triggers and the graphical marker that will be used for each:

Table 9.8 - Intermediate Event Types

Trigger	Description	Marker
None	This is valid for only Intermediate Events that are in the main flow of the Process. The modeler does not display the type of Event. It is used for modeling methodologies that use Events to indicate some change of state in the Process.	
Message	A message arrives from a participant and triggers the Event. This causes the Process to continue if it was waiting for the message, or changes the flow for exception handling. When used to "catch" the message, then the Event marker will be unfilled (see top figure on the right). In Normal Flow, Message Intermediate Events can be used for sending messages to a participant. When used to "throw" the message, the Event marker will be filled (see bottom figure on the right) If used for exception handling it will change the Normal Flow into an Exception Flow.	
Timer	A specific time-date or a specific cycle (e.g., every Monday at 9am) can be set that will trigger the Event. If used within the main flow it acts as a delay mechanism. If used for exception handling it will change the Normal Flow into an Exception Flow.	
Error	This type of Event can only be attached to the boundary of an activity, thus it reacts to (catches) a named error, or to any error if a name is not specified.	
Cancel	This type of Intermediate Event is used within a Transaction Sub-Process. This type of Event MUST be attached to the boundary of a Sub-Process. It SHALL be triggered if a Cancel End Event is reached within the Transaction Sub-Process. It also SHALL be triggered if a Transaction Protocol "Cancel" message has been received while the Transaction is being performed.	

Table 9.8 - Intermediate Event Types

Compensation	This is used for compensation handling--both activating and performing compensation. When used in Normal flow, this Intermediate Event indicates that a Compensation is necessary. Thus, it is used to "throw" the Compensation event, and the Event marker MUST be filled (see the bottom figure on the right). If the Event identifies an activity, then that is the activity (and no other) that will be compensated. Otherwise, the compensation is broadcast to all activities that have completed within the Process Instance, including the top-level Process and including all Sub-Processes. Each completed activity that is subject to compensation will be compensated, in the reverse order of the completion of the activities. To be compensated, an activity MUST have a Compensation Intermediate Event attached to its boundary. When attached to the boundary of an activity, the Event will be triggered by a thrown compensation that identifies that activity or to a broadcast compensation. When used to "catch" the Compensation event, the Event marker MUST be unfilled (see the top figure on the right). When the Event is triggered, the Compensation Activity that is Associated to the Event will be performed (see Figure 9.13).	
Conditional	This type of event is triggered when a Condition becomes true.The attributes of a Condition can be found in Section B.11.5, "Condition," on page 278.	
Link	A Link is a mechanism for connecting two sections of a Process. Link Events can be used to create looping situations or to avoid long Sequence Flow lines. Link Event uses are limited to a single Process level (i.e., they cannot link a parent Process with a Sub-Process). Paired Intermediate Events can also be used as "Off-Page Connectors" for printing a Process across multiple pages. They can also be used as generic "Go To" objects within the Process level. There can be multiple Source Link Events, but there can only be one Target Link Event. When used to "catch" from the Source Link, the Event marker will be unfilled (see the top figure on the right). When used to "throw" to the Target Link, the Event marker will be filled (see the bottom figure on the right).	
Signal	This type of event is used for sending or receiving Signals. A Signal is for general communication within and across Process Levels, across Pools, and between Business Process Diagrams. A BPMN Signal is similar to a signal flare that shot into the sky for anyone who might be interested to notice and then react. Thus, there is a source of the Signal, but no specific intended target. This is different than a BPMN Message, which has a specific Source and a specific Target (which can be an Entity or an abstract Role). This type of Intermediate Event can send or receive a Signal if the Event is part of a Normal Flow. The Event can only receive a Signal when attached to the boundary of an activity. The Signal Event differs from an Error Event in that the Signal defines a more general, non-error condition for interrupting activities (such as the successful completion of another activity) as well as having a larger scope than Error Events. When used to "catch" the signal, the Event marker will be unfilled (see the top figure on the right). When used to "throw" the signal, the Event marker will be filled (see the bottom figure on the right). The attributes of a Signal can be found in Section B.11.17, "Signal," on page 285.	

Table 9.8 - Intermediate Event Types

Multiple	This means that there are multiple Triggers assigned to the Event. If used within normal flow, the Event can "catch" the Trigger or "throw" the Triggers. When attached to the boundary of an activity, the Event can only "catch" the Trigger. When used to "catch" the Trigger, only one of the assigned Triggers is required and the Event marker will be unfilled (see the top figure on the right). When used to "throw" the Trigger (the same as a Multiple End Event), all the assigned Triggers will be thrown and the Event marker will be filled (see the bottom figure on the right).	

Attributes

The following table displays the set of attributes of an Intermediate Event, which extends the set of common Event attributes (see Table 9.9):

Table 9.9 - Intermediate Event Attributes

Attributes	Description
Trigger (0-n) : EventDetail	Trigger (EventDetail) is an attribute that defines the type of trigger expected for an Intermediate Event. Of the set of EventDetailTypes (see Section 9.3.5, "Event Details," on page 49), only eight (8) can be applied to an Intermediate Event: Message, Timer, Error, Cancel, Compensation, Conditional, Link, and Signal. (see Table 9.8). If no EventDetail is defined, then this is considered a None Intermediate Event and the Event will not have an internal marker (see Table 9.8). If more than one EventDetail is defined, this is considered a Multiple Intermediate Event and the Event will have the star internal marker (see Table 9.8).
Target (0-1) : Activity	A Target MAY be included for the Intermediate Event. The Target MUST be an activity (Sub-Process or Task). This means that the Intermediate Event is attached to the boundary of the activity and is used to signify an exception or compensation for that activity.

Activity Boundary Connections

An Intermediate Event can be attached to the boundary of an activity under the following conditions:

◆ (One or more) Intermediate Events MAY be attached directly to the boundary of an Activity.

  ◆ To be attached to the boundary of an Activity, an Intermediate Event MUST be one of the following Triggers: Message, Timer, Error, Cancel, Compensation, Conditional, Signal, and Multiple.

    ◆ An Intermediate Event with a Cancel Trigger MAY be attached to a Sub-Process boundary only if the Transaction attribute of the Sub-Process is set to TRUE.

Sequence Flow Connections

See Section 8.4.1, "Sequence Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Sequence Flow.

◆ The following Intermediate Events MAY be attached to the boundary of an Activity: Message, Timer, Error, Cancel (only Sub-Process that is a Transaction), Compensation, Conditional, Signal, and Multiple. Thus, the following MUST NOT: None, and Link.

- ◆ If the Intermediate Event is attached to the boundary of an activity:
    - ◆ The Intermediate Event MUST NOT be a target for Sequence Flow; it cannot have an incoming Flow.
    - ◆ The Intermediate Event MUST be a source for Sequence Flow; it can have one (and only one) outgoing Sequence Flow.
        - ◆ An exception to this: an Intermediate Event with a Compensation Trigger MUST NOT have an outgoing Sequence Flow (it MAY have an outgoing Association).
- ◆ The following Intermediate Events MAY be used in Normal Flow: None, Message, Timer, Compensation, Conditional, Link, and Signal. Thus, the following MUST NOT: Cancel, Error, and Multiple.
    - ◆ If the Intermediate Event is used within Normal Flow:
        - ◆ Intermediate Events of the following types MUST be a target of a Sequence Flow: None, and Compensation. They MUST have one (and only one) incoming Flow.
        - ◆ Intermediate Events of the following types MAY be a target of a Sequence Flow: Message, Timer, Conditional, Link, and Signal. They MAY have one (and only one) incoming Flow.

Note – These types of Intermediate Events will always be ready to accept the Event Triggers (once) while the Process in which they are contained is active. They are NOT optional and are expected to be triggered during the performance of the Process.

- ◆ An Intermediate Event MUST be a source for Sequence Flow; it MUST have one (and only one) outgoing Sequence Flow.
    - ◆ An exception to this: an Source Link Intermediate Event (as defined below), it is not required to have an outgoing Sequence Flow.
- ◆ An Intermediate Event with a Link Trigger MUST NOT be both a target and a source of a Sequence Flow.

To define the use of a Link Intermediate Event as an "Off-Page Connector" or a "Go To" object:

- ◆ A Link Intermediate Event MAY be the target (Target Link) or a source (Source Link) of a Sequence Flow, but MUST NOT be both a target and a source.
    - ◆ If there is a Source Link, there MUST be a matching Target Link (they have the same Name).
        - ◆ There MAY be multiple Source Links for a single Target Link.
        - ◆ There MUST NOT be multiple Target Links for a single Source Link.

Message Flow Connections

See Section 8.4.2, "Message Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Message Flow.

Note – All Message Flow must connect two separate Pools. They can connect to the Pool boundary or to Flow Objects within the Pool boundary. They cannot connect two objects within the same Pool.

- ◆ An Intermediate Event of type Message MAY be the target for Message Flow; it can have one incoming Message Flow.
- ◆ An Intermediate Event of type Message MAY be a source for Message Flow; it can have one outgoing Message Flow.
- ◆ An Intermediate Event of type Message MAY have an incoming Message Flow or an outgoing Message Flow, but

not both.

9.3.5 Event Details

Event Details refers to the Triggers of Start and Intermediate Events and the Results of End Events. The types of Event Details are: Message, Timer, Error, Cancel, Compensation, Conditional, Link, Signal, and Terminate. A None Event is determined by an Event that does not specify an Event Detail. A Multiple Event is determined by an Event that specifies more than one Event Detail. The different types of Events, (Start, Intermediate, and End) utilize a subset of the available types of Event Details (see Figure 9.5).

Figure 9.5 - **Event Details as Applied to Start, Intermediate, and End Events**

The following sections will present the attributes common to all Event Details and the specific attributes for the Event Details that have additional attributes. Note that the Cancel and Terminate Event Details do not have additional attributes.

Common Event Detail Attributes

The following table displays the set of attributes common to the types of EventDetail, and which extends the set of common BPMN element attributes (see Table 9.1).

Table 9.10 - Common Event Detail Attributes

Attributes	Description
EventDetailType (Message | Timer | Error | Conditional | Link | Signal | Compensate | Cancel | Terminate) Message : String	The EventDetailType attribute defines the type of trigger expected for an Event. The set of types includes Message, Timer, Error, Conditional, Link, Signal, Compensate, Cancel, and Terminate. The EventTypes (Start, Intermediate, and End) will each have a subset of the EventDetailTypes that can be used. The EventDetailType list MAY be extended to include new types. These new types MAY have a new modeler- or tool-defined Marker to fit within the boundaries of the Event.

Conditional Event Detail

The following table displays the set of attributes a Conditional EventDetail, and which extends the set of common Event Detail attributes (see Table 9.10).

Table 9.11 - Conditional EventDetail Attributes

Attributes	Description
ConditionRef : Condition	If the Trigger is Conditional, then a Condition MUST be entered. The attributes of a Condition can be found in Section B.11.5, "Condition," on page 278.

Compensation Event Detail

The following table displays the set of attributes a Compensation EventDetail, and which extends the set of common Event Detail attributes (see Table 9.10).

Table 9.12 - Compensation EventDetail Attributes

Attributes	Description
ActivityRef (0-1): Activity	*For an End Event:* If the Result is a Compensation, then the Activity to be compensated MAY be supplied. If an Activity is not supplied, then the Event is broadcast to all completed activities in the Process Instance. *For an Intermediate Event within Normal Flow:* If the Trigger is a Compensation, then the Activity to be compensated MAY be supplied. If an Activity is not supplied, then the Event is broadcast to all completed activities in the Process Instance. This "throws" the compensation. *For an Intermediate Event attached to the boundary of an Activity:* This Event "catches" the compensation. No further information is required. The Activity the Event is attached to will provide the Id necessary to match the compensation event with the event that "threw" the compensation or the compensation will be a broadcast.

Error Event Detail

The following table displays the set of attributes a Error EventDetail, and which extends the set of common Event Detail attributes (see Table 9.10).

Table 9.13 - Error EventDetail Attributes

Attributes	Description
ErrorCode : String	*For an End Event:* If the Result is an Error, then the ErrorCode MUST be supplied.This "throws" the error. *For an Intermediate Event within Normal Flow:* If the Trigger is an Error, then the ErrorCode MUST be entered. This "throws" the error. *For an Intermediate Event attached to the boundary of an Activity:* If the Trigger is an Error, then the ErrorCode MAY be entered. This Event "catches" the error. If there is no ErrorCode, then any error SHALL trigger the Event. If there is an ErrorCode, then only an error that matches the ErrorCode SHALL trigger the Event.

Link Event Detail

The following table displays the set of attributes a Link EventDetail, and which extends the set of common Event Detail attributes (see Table 9.10).

Table 9.14 - Link EventDetail Attributes

Attributes	Description
Name : String	If the Trigger is a Link, then the Name MUST be entered.

Message Event Detail

The following table displays the set of attributes a Message EventDetail, and which extends the set of common Event Detail attributes (see Table 9.10).

Table 9.15 - Message EventDetail Attributes

Attributes	Description
MessageRef : Message	If the EventDetailType is a MessageRef, then the a Message MUST be supplied. The attributes of a Message can be found in Section B.11.11, "Message," on page 283.
Implementation (Web Service | Other | Unspecified) Web Service	This attribute specifies the technology that will be used to send or receive the message. A Web service is the default technology.

Signal Event Detail

The following table displays the set of attributes a Signal EventDetail, and which extends the set of common Event Detail attributes (see Table 9.10).

Table B.16 - Signal EventDetail Attributes

Attributes	Description
SignalRef : String	If the Trigger is a Signal, then a Signal Shall be entered. The attributes of a Signal can be found in Section B.11.17, "Signal," on page 285.

Timer Event Detail

The following table displays the set of attributes a Timer EventDetail, and which extends the set of common Event Detail attributes (see Table 9.10).

Table 9.17 - Timer EventDetail Attributes

Attributes	Description
TimeDate (0-1) : TimeDateExpression	If the Trigger is a Timer, then a TimeDate MAY be entered. If a TimeDate is not entered, then a TimeCycle MUST be entered (see the attribute below). The attributes of a TimeDateExpression can be found in Section B.11.18, "TimeDateExpression," on page 285
TimeCycle (0-1) : TimeDateExpression	If the Trigger is a Timer, then a TimeCycle MAY be entered. If a TimeCycle is not entered, then a TimeDate MUST be entered (see the attribute above).

9.4   Activities

An activity is work that is performed within a business process. An activity can be atomic or non-atomic (compound). The types of activities that are a part of a Business Process Diagram are: Process, Sub-Process, and Task. However, a Process is not a specific graphical object. Instead, it is a set of graphical objects. The following sections will focus on the graphical objects Sub-Process and Task. More information about Processes can be found in Section 8.6, "Processes," on page 32.

9.4.1 Common Activity Attributes

The following table displays the set of attributes common to both a Sub-Process and a Task, and which extends the set of common Flow Object attributes (see Table 9.3) -- Note that Table 9.19 and Table 9.20 contain additional attributes that must be included within this set if extended by any other attribute table:

Table 9.18 - Common Activity Attributes

Attributes	Description
ActivityType (Task | Sub-Process) Task : String	The ActivityType MUST be of type Task or Sub-Process.
Status (None | Ready | Active | Cancelled | Aborting | Aborted | Completing | Completed) None : String	The Status of an activity is determined when the activity is being executed by a process engine. The Status of an activity can be used within Assignment Expressions.
Performers (0-n) : String	One or more Performers MAY be entered. The Performer attribute defines the resource that will perform or will be responsible for the activity. The Performer entry could be in the form of a specific individual, a group, an organization role or position, or an organization.

Table 9.18 - Common Activity Attributes

Attributes	Description
Properties (0-n) : Property	Modeler-defined Properties MAY be added to an activity. These Properties are "local" to the activity. These Properties are only for use within the processing of the activity. The fully delineated name of these properties is "<process name>.<activity name>.<property name>" (e.g., "Add Customer.Review Credit.Status"). Further details about the definition of a Property can be found in Section B.11.15, "Property," on page 285.
InputSets (0-n) : InputSet	The InputSets attribute defines the data requirements for input to the activity. Zero or more InputSets MAY be defined. Each InputSet is sufficient to allow the activity to be performed (if it has first been instantiated by the appropriate signal arriving from an incoming Sequence Flow). Further details about the definition of an InputSet can be found in Section B.11.10, "InputSet," on page 282.
OutputSets (0-n) : OutputSet	The OutputSets attribute defines the data requirements for output from the activity. Zero or more OutputSets MAY be defined. At the completion of the activity, only one of the OutputSets may be produced--It is up to the implementation of the activity to determine which set will be produced. However, the IORules attribute MAY indicate a relationship between an OutputSet and an InputSet that started the activity. Further details about the definition of an OutputSet can be found in Section B.11.13, "OutputSet," on page 284.
IORules (0-n) : Expression	The IORules attribute is a collection of expressions, each of which specifies the required relationship between one input and one output. That is, if the activity is instantiated with a specified input, that activity shall complete with the specified output.
StartQuantity 1 : Integer	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of Tokens that must arrive before the activity can begin.
CompletionQuantity 1 : Integer	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of Tokens that must be generated from the activity. This number of Tokens will be sent done any outgoing Sequence Flow (assuming any Sequence Flow Conditions are satisfied).
LoopType (None | Standard | MultiInstance) None : String	LoopType is an attribute and is by default None, but MAY be set to Standard or MultiInstance. If so, the Loop marker SHALL be placed at the bottom center of the activity shape (see Figure 9.9 and Figure 9.15). A Task of type Receive that has its Instantiate attribute set to True MUST NOT have a Standard or MultiInstance LoopType.

Standard Loop Attributes

A Standard Loop activity will have a boolean expression that is evaluated after each cycle of the loop. If the expression is still True, then the loop will continue. There are two variations of the loop, which reflect the programming constructs of while and until. That is, a while loop will evaluate the expression <u>before</u> the activity is performed, which means that the activity may not actually be performed. The until loop will evaluate the expression <u>after</u> the activity has been performed, which means that the activity will be performed at least once.

The following are additional attributes of a Standard Loop Activity (where the LoopType attribute is set to "Standard"), which extends the set of common activity attributes (see Table 9.18):

Table 9.19 - Standard Loop Activity Attributes

Attributes	Description
LoopCondition : Expression	Standard Loops MUST have a boolean Expression to be evaluated, plus the timing when the expression SHALL be evaluated. The attributes of an Expression can be found in Section B.11.8, "Expression," on page 281.
LoopCounter : Integer	The LoopCounter attribute is used at runtime to count the number of loops and is automatically updated by the process engine. The LoopCounter attribute MUST be incremented at the start of a loop. The modeler may use the attribute in the LoopCondition Expression.
LoopMaximum (0-1) : Integer	The Maximum an optional attribute that provides is a simple way to add a cap to the number of loops. This SHALL be added to the Expression defined in the LoopCondition.
TestTime (Before | After) After : String	The expressions that are evaluated Before the activity begins are equivalent to a programming while function. The expression that are evaluated After the activity finishes are equivalent to a programming until function.

Multi-Instance Loop Attributes

Multi-Instance loops reflect the programming construct for each. The loop expression for a Multi-Instance loop is a numeric expression evaluated only once before the activity is performed. The result of the expression evaluation will be an integer that will specify the number of times that the activity will be repeated.

There are also two variations of the Multi-Instance loop where the instances are either performed sequentially or in parallel.

The following are additional attributes of a Multi-Instance Loop Activity (where the LoopType attribute is set to "MultiInstance"), which extends the set of common activity attributes (see Table 9.18):

Table 9.20 - Multi-Instance Loop Activity Attributes

Attributes	Description
MI_Condition : Expression	MultiInstance Loops MUST have a numeric Expression to be evaluated--the Expression MUST resolve to an integer. The attributes of an Expression can be found in Section B.11.8, "Expression," on page 281.
LoopCounter : Integer	The LoopCounter attribute is only applied for Sequential MultiInstance Loops and for processes that are being executed by a process engine. The attribute is updated at runtime by a process engine to count the number of loops as they occur. The LoopCounter attribute MUST be incremented at the start of a loop. Unlike a Standard loop, the modeler does not use this attribute in the MI_Condition Expression, but it can be used for tracking the status of a loop.

Table 9.20 - Multi-Instance Loop Activity Attributes

Attributes	Description
MI_Ordering (Sequential | Parallel) Sequential : String	This applies to only MultiInstance Loops. The MI_Ordering attribute defines whether the loop instances will be performed sequentially or in parallel. Sequential MI_Ordering is a more traditional loop. Parallel MI_Ordering is equivalent to multi-instance specifications that other notations, such as UML Activity Diagrams use. If set to Parallel, the Parallel marker SHALL replace the Loop Marker at the bottom center of the activity shape (see Figure 9.9 and Table 9.18).
[Parallel MI_Ordering only] **MI_FlowCondition** (None | One | All | Complex) All : String	This attribute is equivalent to using a Gateway to control the flow past a set of parallel paths. • An MI_FlowCondition of "None" is the same as uncontrolled flow (no Gateway) and means that all activity instances SHALL generate a token that will continue when that instance is completed. • An MI_FlowCondition of "One" is the same as an Exclusive Gateway and means that the Token SHALL continue past the activity after only one of the activity instances has completed. The activity will continue its other instances, but additional Tokens MUST NOT be passed from the activity. • An MI_FlowCondition of "All" is the same as a Parallel Gateway and means that the Token SHALL continue past the activity after all of the activity instances have completed. • An MI_FlowCondition of "Complex" is similar to that of a Complex Gateway. The ComplexMI_FlowCondition attribute will determine the Token flow.
[Complex MI_FlowCondition only] **ComplexMI_FlowCondition** (0-1) : Expression	If the MI_FlowCondition attribute is set to "Complex," then an Expression Must be entered. This expression that MAY reference Process data. The expression will be evaluated after each iteration of the Activity and SHALL resolve to a boolean. If the result of the expression evaluation is TRUE, then a Token will be sent down the activity's outgoing Sequence Flow. Otherwise, no Token for that iteration will be sent. The attributes of an Expression can be found in the Section B.11.8, "Expression," on page 281.

9.4.2 Sub-Process

A **Sub-Process** is a compound activity in that it has detail that is defined as a flow of other activities. A Sub-Process is a graphical object within a Process Flow, but it also can be "opened up" to show another Process (either Embedded or Reusable). A Sub-Process object shares the same shape as the Task object, which is a rounded rectangle.

◆ A Sub-Process is a rounded corner rectangle that MUST be drawn with a single thin black line.

    ◆ The use of text, color, size, and lines for a Sub-Process MUST follow the rules defined in Section 8.3, "Use of Text, Color, Size, and Lines in a Diagram," on page 29 with the exception that.

◆ The boundary drawn with a double line SHALL be reserved for Sub-Process that has its IsATransaction attribute set to True**.**

The Sub-Process can be in a collapsed view that hides its details (see Figure 9.6) or a Sub-Process can be in an expanded view that shows its details within the view of the Process in which it is contained (see Figure 9.7). In the collapsed form, the Sub-Process object uses a marker to distinguish it as a Sub-Process, rather than a Task.

◆ The Sub-Process marker MUST be a small square with a plus sign (+) inside. The square MUST be positioned at the bottom center of the shape.
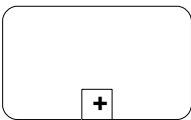
Figure 9.6 - Collapsed Sub-Process
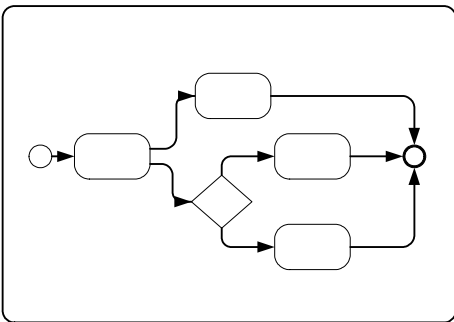
Figure 9.7 - Expanded Sub-Process

Expanded Sub-Process may be used for multiple purposes. They can be used to "flatten" a hierarchical process so that all detail can be shown at the same time. They are used to create a context for exception handling that applies to a group of activities (Section 10.2.2, "Exception Flow," on page 131 for more details). Compensations can be handled similarly (Section 10.3, "Compensation Association," on page 133 for more details).

Expanded Sub-Process may be used as a mechanism for showing a group of parallel activities in a less-cluttered, more compact way. In Figure 9.8, activities "C" and "D" are enclosed in an unlabeled Expanded Sub-Process. These two activities will be performed in parallel. Notice that the Expanded Sub-Process does not include a Start Event or an End Event and the Sequence Flow to/from these Events. This usage of Expanded Sub-Processes for "parallel boxes" is the motivation for having Start and End Events being optional objects.
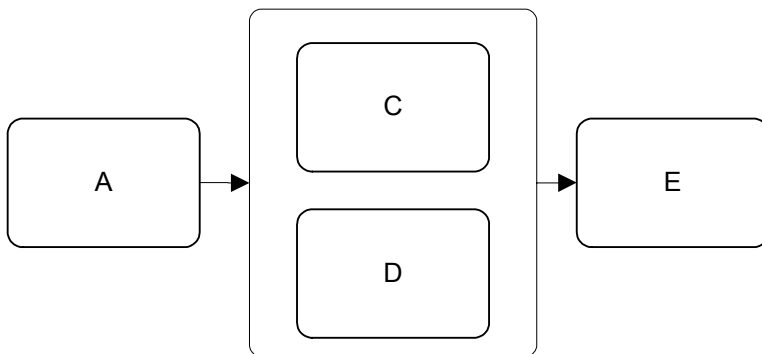
Figure 9.8 - Expanded Sub-Process used as a "parallel box"

BPMN specifies five types of standard markers for Sub-Processes. The (Collapsed) Sub-Process Marker, seen in Figure 9.6, can be combined with four other markers: a Loop Marker or a Parallel Marker, a Compensation Marker, and an Ad Hoc Marker. A collapsed Sub-Process may have one to three of these other markers, in all combinations except that Loop and Multiple Instance cannot be shown at the same time (see Figure 9.9).

- ◆ The marker for a Sub-Process that loops MUST be a small line with an arrowhead that curls back upon itself.
  - ◆ The Loop Marker MAY be used in combination with any of the other markers except the Multiple Instance Marker.
- ◆ The marker for a Sub-Process that has multiple instances MUST be set of three vertical lines in parallel.
  - ◆ The Multiple Instance Marker MAY be used in combination with any of the other markers except the Loop Marker.
- ◆ The marker for a Sub-Process that is Ad Hoc MUST be a "tilde" symbol.
  - ◆ The Ad-Hoc Marker MAY be used in combination with any of the other markers.
- ◆ The marker for a Sub-Process that is used for compensation MUST be a pair of left facing triangles (like a tape player "rewind" button).
  - ◆ The Compensation Marker MAY be used in combination with any of the other markers.
- ◆ All the markers that are present MUST be grouped and the whole group centered at the bottom of the shape.
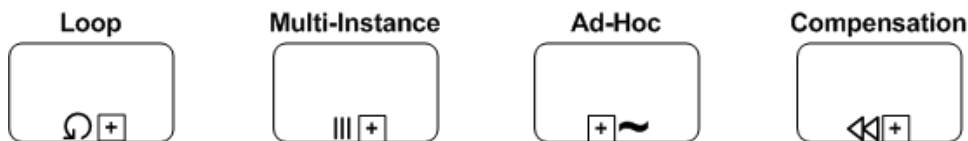
Figure 9.9 - Collapsed Sub-Process Markers

Attributes

The following table displays the set of attributes of a Sub-Process, which extends the set of common activity attributes (see Table 9.21):

Table 9.21 - Sub-Process Attributes

Attributes	Description
SubProcessType (Embedded | Reusable | Reference) Embedded : String	SubProcessType is an attribute that defines whether the Sub-Process details are embedded within the higher level Process or refers to another, re-usable Process. The default is Embedded.
IsATransaction False : Boolean	IsATransaction determines whether or not the behavior of the Sub-Process will follow the behavior of a Transaction (see Section , "Sub-Process Behavior as a Transaction," on page 63).
TransactionRef (0-1) : Transaction	If the IsATransaction attribute is False, then a Transaction MUST NOT be identified. If the IsATransaction attribute is True, then a Transaction MUST be identified. The attributes of a Transaction can be found in the Section B.11.19, "Transaction," on page 286. Note that Transactions that are in different Pools and are connected through Message Flow MUST have the same TransactionId.

Embedded Sub-Process

An Embedded (or nested) Sub-Process object is an activity that contains other activities (a Process). The Process within the Process is dependent on the parent Process for instigation and has visibility to the parent's global data. No mapping of data is required.

The objects within the Embedded Sub-Process, being dependent on their parent, do not have all the features of a full Business Process Diagram, such as Pools and Lanes. Thus, an expanded view of the Embedded Sub-Process would only contain Flow Objects, Connecting Objects, and Artifacts (see Figure 9.8).

• All Start Events for an Embedded Sub-Process MUST be of type None.

The following are additional attributes of a Embedded Sub-Process (where the SubProcessType attribute is set to "Embedded"), which extends the set of Sub-Process attributes (see Table 9.22):

Table 9.22 - Embedded Sub-Process Attributes

Attributes	Description
GraphicalElements (0-n) : Object	The GraphicalElements attribute identifies all of the objects (e.g., Events, Activities, Gateways, and Artifacts) that are contained within the Embedded Sub-Process.
AdHoc False : Boolean	AdHoc is a boolean attribute, which has a default of False. This specifies whether the Embedded Sub-Process is Ad Hoc or not. The activities within an Ad Hoc Embedded Sub-Process are not controlled or sequenced in a particular order, there performance is determined by the performers of the activities.

Table 9.22 - Embedded Sub-Process Attributes

Attributes	Description
[AdHoc = True only] **AdHocOrdering** (0-1) (Sequential | Parallel) Parallel : String	If the Embedded Sub-Process is Ad Hoc (the AdHoc attribute is True), then the AdHocOrdering attribute MUST be included. This attribute defines if the activities within the Process can be performed in Parallel or must be performed sequentially. The default setting is Parallel and the setting of Sequential is a restriction on the performance that may be required due to shared resources.
[AdHoc = True only] **AdHocCompletionCondition** (0-1) : Expression	If the Embedded Sub-Process is Ad Hoc (the AdHoc attribute is True), then a Completion Condition MUST be included, which defines the conditions when the Process will end. The Ad Hoc marker SHALL be placed at the bottom center of the Process or the Sub-Process shape for Ad Hoc Processes.

Reusable Sub-Process

An Reusable Sub-Process object is an activity within a Process that "calls" to another Process that exists within a BDP (see Figure 9.10). The Process that is called is not dependent on the Reusable Sub-Process object's parent Process for global data. The Reusable Sub-Process object may pass data to/from the called Process.
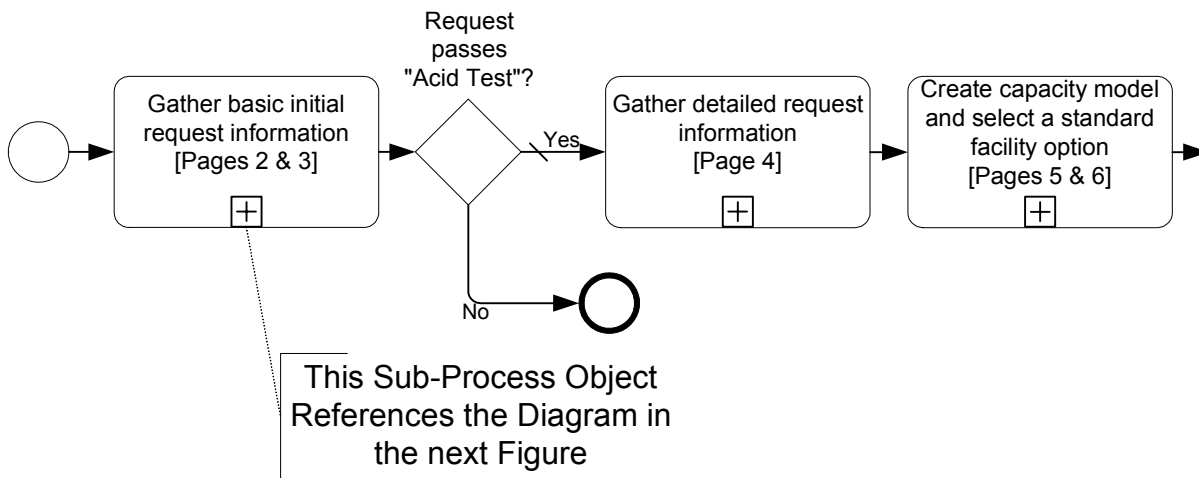
Figure 9.10 - A Sub-Process Object with its Details Shown in the diagram of the next Figure

The called Process will exist in a separate diagram that can have multiple Pools. Any view of the called Process (including an expanded view within the calling Process) would show the whole diagram in which the called Process resides (see Figure 9.11), but any data mapping will be only to that Process and not to any of the other Processes that might be in the called diagram.
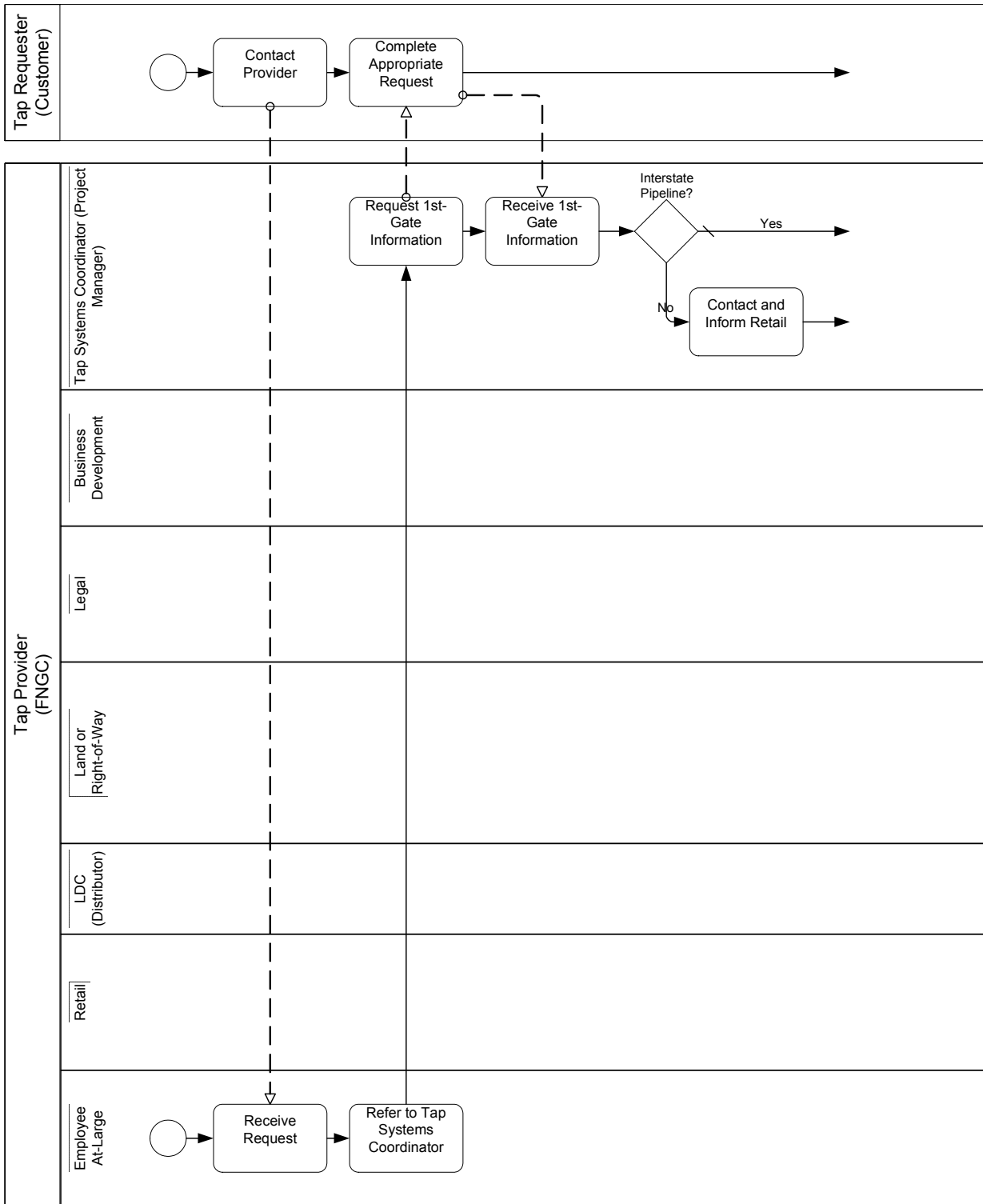
Figure 9.11 - A Process and Diagram Details of the Sub-Process Object in the Previous Figure

The called Process will (MUST) be instantiated as a Sub-Process through a None Start Event. Being reusable, the Process could also be instantiated as a Sub-Process by other Independent Sub-Process objects (in the same or other diagrams). In addition, it can be instantiated as a top-level Process through a separate Start Event that has a Trigger (other than None-- see Figure 9.12).
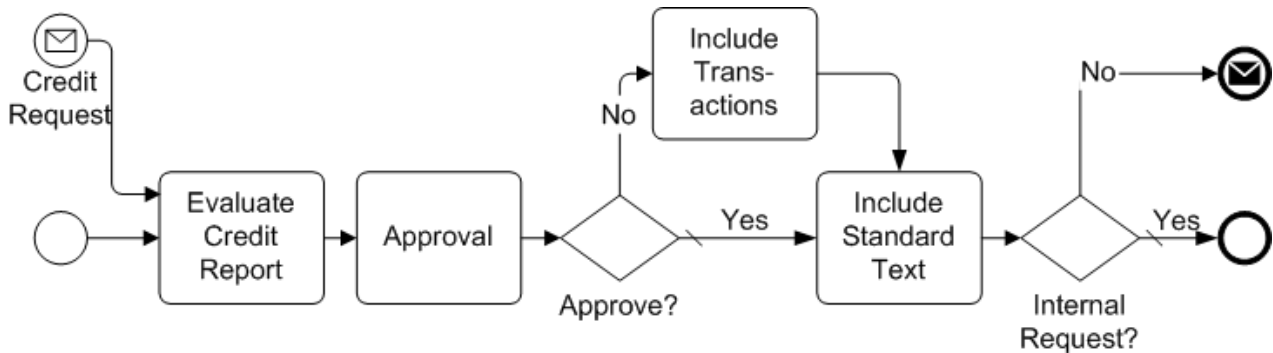
Figure 9.12 - A Process that is used as a Sub-Process or a Top-Level Process

The following are additional attributes of a Reusable Sub-Process (where the SubProcessType attribute is set to "Reusable"), which extends the set of Sub-Process attributes (see Table 9.23):

Table 9.23 - Reusable Sub-Process Attributes

Attributes	Description
DiagramRef: Business Process Diagram	The BPD MUST be identified. The attributes of a BPD can be found in the Section 8.5, "Business Process Diagram Attributes," on page 31.
ProcessRef: Process	A Process MUST be identified. The attributes of a Process can be found in the Section 8.6, "Processes," on page 32.
InputMaps (0-n) : Expression	Multiple input mappings MAY be made between the Reusable Sub-Process and the Process referenced by this object. These mappings are in the form of an expression. A specific mapping expression MUST specify the mapping of Properties between the two Processes OR the mapping of Artifacts between the two Processes.
OutputMaps (0-n) : Expression	Multiple output mappings MAY be made between the Reusable Sub-Process and the Process referenced by this object. These mappings are in the form of an expression. A specific mapping expression MUST specify the mapping of Properties between the two Processes OR the mapping of Artifacts between the two Processes.

Reference Sub-Process

There may be times where a modeler may want to reference another Sub-Process that has been defined. If the two Sub-Processes share the exact same behavior and properties, then by one referencing the other, the attributes that define the behavior only have to be created once and maintained in only one location.

The following table displays the set of attributes of a Reference Sub-Process (where the SubProcessType attribute is set to "Reference"), which extends the set of Sub-Process attributes (see Table 9.24):

Table 9.24 - Reference Sub-Process Attributes

Attributes	Description
SubProcessRef : Sub-Process	The Sub-Process being referenced MUST be identified. The attributes for the Sub-Process element can be found in Table 9.21.

Sub-Process Behavior as a Transaction

A Sub-Process, either collapsed or expanded, can be set as being a Transaction, which will have a special behavior that is controlled through a transaction protocol (such as BTP or WS-Transaction). The boundary of the activity will be double-lined to indicate that it is a Transaction (see Figure 9.13).
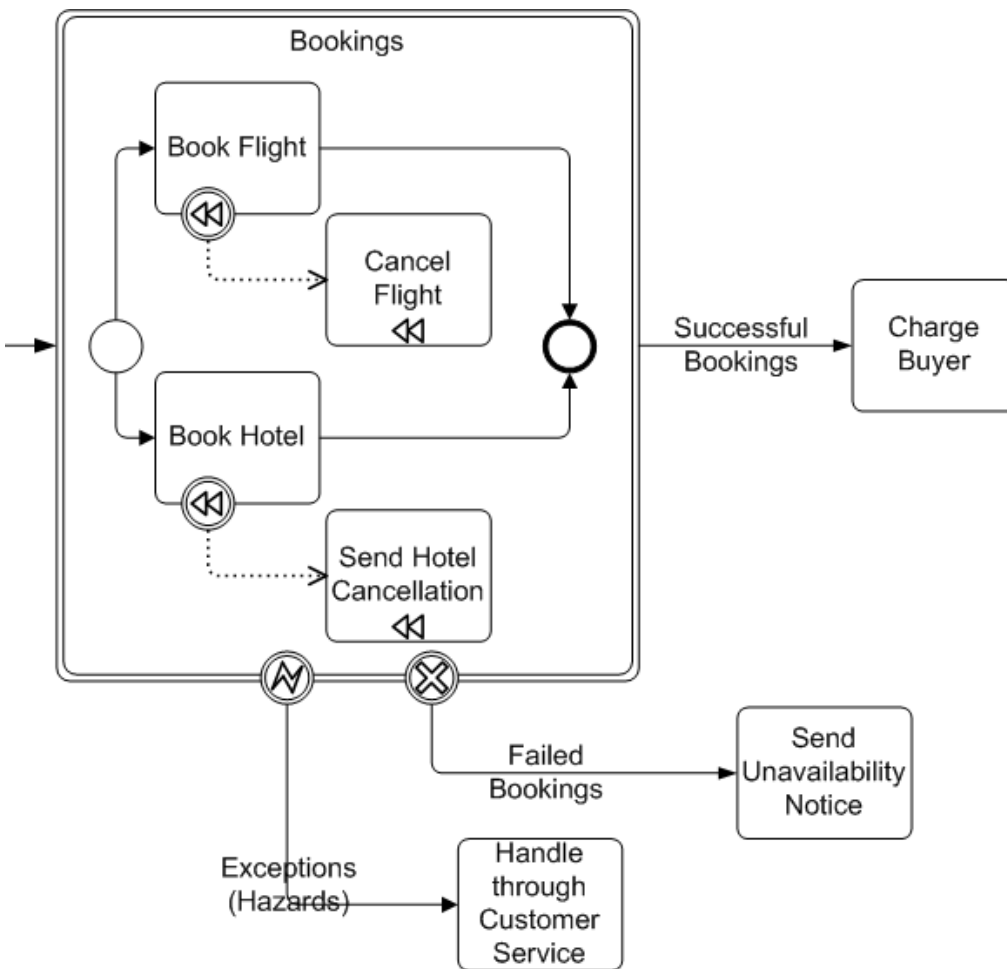
Figure 9.13 - An Example of a Transaction Expanded Sub-Process

There are three basic outcomes of a Transaction:

- Successful completion: this will be shown as a normal Sequence Flow that leaves the Sub-Process.

- Failed completion (Cancel): When a Transaction is cancelled, then the activities inside the Transaction will be subjected to the cancellation actions, which could include rolling back the process and compensation for specific activities. Note that other mechanisms for interrupting a Sub-Process will not cause Compensation (e.g., Error, Timer, and anything for a non-Transaction activity). A Cancel Intermediate Event, attached to the boundary of the activity, will direct the flow after the Transaction has been rolled back and all compensation has been completed. The Cancel Intermediate Event can only be used when attached to the boundary of a Transaction activity. It cannot be used in any Normal Flow and cannot be attached to a non-Transaction activity. There are two mechanisms that can signal the cancellation of a Transaction:

  - A Cancel End Event is reached within the Transaction Sub-Process. A Cancel End Event can only be used within a Sub-Process that is set to a Transaction.

  - A Cancel Message can be received via the Transaction Protocol that is supporting the execution of the Sub-Process.

- Hazard: This means that something went terribly wrong and that a normal success or cancel is not possible. We are using an Error to show Hazards. When a Hazard happens, the activity is interrupted (without Compensation) and the flow will continue from the Error Intermediate Event.

The behavior at the end of a successful Transaction Sub-Process is slightly different than that of a normal Sub-Process. When each path of the Transaction Sub-Process reaches a non-Cancel End Event(s), the flow does not immediately move back up to the higher-level Parent Process, as does a normal Sub-Process. First, the transaction protocol must verify that all the participants have successfully completed their end of the Transaction. Most of the time this will be true and the flow will then move up to the higher-level Process. But it is possible that one of the participants may end up with a problem that causes a Cancel or a Hazard. In this case, the flow will then move to the appropriate Intermediate Event, even though it had apparently finished successfully.

Sequence Flow Connections

See Section 8.4.1, "Sequence Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Sequence Flow.

- ◆ A Sub-Process MAY be a target for Sequence Flow; it can have multiple incoming Flow. Incoming Flow MAY be from an alternative path and/or a parallel paths.

  - ◆ The Incoming Sequence Flow's attribute TargetRef MAY be extended to include both the Sub-Process object (at the parent level) and a Start Event that resides within the details of the Sub-Process. This provides a direct connection from the parent-level Sequence Flow to the lower-level Start Event for situations where there is more than one Start Event in the Sub-Process. The form of the extension would be "Sub-Process.Start."

    - ◆ If the details of the Sub-Process (i.e., it's Start Events) are not visible or accessible to the modeler, then the determination as to which Start Event, if there are multiple, will be triggered is undefined. But only one of the Start Events will be triggered.

Note – If the Sub-Process has multiple incoming Sequence Flow, then this is considered uncontrolled flow. This means that when a Token arrives from one of the Paths, the Sub-Process will be instantiated. It will not wait for the arrival of Tokens from the other paths. If another Token arrives from the same path or another path, then a separate instance of the Sub-Process will be created. If the flow needs to be controlled, then the flow should converge on a Gateway that precedes the Sub-Process (Section 9.5, "Gateways," on page 71 for more information on Gateways).

- ◆ If the Sub-Process does not have an incoming Sequence Flow, and there is no Start Event for the Process, then the

Sub-Process MUST be instantiated when the process is instantiated.

◆ Exceptions to this are Sub-Processes that are defined as being Compensation activities (have the Compensation Marker). Compensation Sub-Processes are not considered a part of the Normal Flow and MUST NOT be instantiated when the Process is instantiated.

◆ A Sub-Process MAY be a source for Sequence Flow; it can have multiple outgoing Flow. If there are multiple outgoing Sequence Flow, then this means that a separate parallel path is being created for each Flow.

Tokens will be generated for each outgoing Sequence Flow from Sub-Process. The TokenIds for each of the Tokens will be set such that it can be identified that the Tokens are all from the same parallel Fork as well as the number of Tokens that exist in parallel.

◆ If the Sub-Process does not have an outgoing Sequence Flow, and there is no End Event for the Process, then the Sub-Process marks the end of one or more paths in the Process. When the Sub-Process ends and there are no other parallel paths active, then the Process MUST be completed.

◆ Exceptions to this are Sub-Processes that are defined as being Compensation activities (have the Compensation Marker). Compensation Sub-Processes are not considered a part of the Normal Flow and MUST NOT mark the end of the Process.

Message Flow Connections

Section 8.4.2, "Message Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Message Flow.

Note – All Message Flow must connect two separate Pools. They can connect to the Pool boundary or to Flow Objects within the Pool boundary. They cannot connect two objects within the same Pool.

◆ A Sub-Process MAY be the target for Message Flow; it can have zero or more incoming Message Flow.

◆ A Sub-Process MAY be a source for Message Flow; it can have zero or more outgoing Message Flow.

9.4.3 Task

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. Generally, an end-user and/or an application are used to perform the Task when it is executed.

A Task object shares the same shape as the Sub-Process, which is a rectangle that has rounded corners (see Figure 9.14).

◆ A Task is a rounded corner rectangle that MUST be drawn with a single thin black line.

◆ The use of text, color, size, and lines for a Task MUST follow the rules defined in Section 8.3, "Use of Text, Color, Size, and Lines in a Diagram," on page 29.

Figure 9.14 - A Task Object

BPMN specifies three types of markers for Task: a Loop Marker or a Multiple Instance Marker and a Compensation Marker. A Task may have one or two of these markers (see Figure 9.15).

- The marker for a Task that is a standard loop MUST be a small line with an arrowhead that curls back upon itself.
    - The Loop Marker MAY be used in combination with the Compensation Marker.
- The marker for a Task that is a multi-instance loop MUST be a set of three vertical lines in parallel.
    - The Multiple Instance Marker MAY be used in combination with the Compensation Marker.
- The marker for a Task that is used for compensation MUST be a pair of left facing triangles (like a tape player "rewind" button).
    - The Compensation Marker MAY be used in combination with the Loop Marker or the Multiple Instance Marker.
- All the markers that are present MUST be grouped and the whole group centered at the bottom of the shape.

All the markers that are present will be grouped and centered at the bottom of the shape.
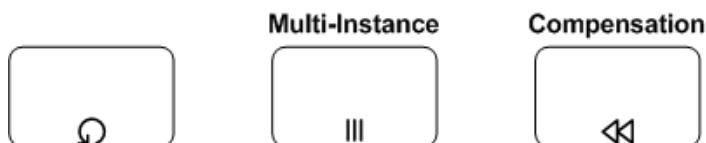
Figure 9.15 - Task Markers

In addition to categories of Task shown above, there are different types of Tasks identified within BPMN to separate the types of inherent behavior that Tasks might represent (see Table 9.2). However, BPMN does not specify any graphical indicators for the different types of Tasks. Modelers or modeling tools may choose to create their own indicators or markers to show the readers of the diagram the type of Task. This is permitted by BPMN as long as the basic shape of the Task (a rounded rectangle) is not modified. The list of Task types may be extended along with any corresponding indicators.

Attributes

The following table displays the set of attributes of a Task, which extends the set of common activity attributes (see Table 9.18):

Table 9.25 - Task Attributes

Attributes	Description
TaskType (Service | Receive | Send | User | Script | Manual | Reference | None) None : String	TaskType is an attribute that has a default of None, but MAY be set to Send, Receive, User, Script, Manual, Reference, or Service. The TaskType will be impacted by the Message Flow to and/or from the Task, if Message Flow are used. A TaskType of Receive MUST NOT have an outgoing Message Flow. A TaskType of Send MUST NOT have an incoming Message Flow. A TaskType of Script or Manual MUST NOT have an incoming or an outgoing Message Flow. The TaskType list MAY be extended to include new types. The attributes for specific values of TaskType can be found in Table 9.26 through Table 9.31.

Service Task

A Service Task is a Task that provides some sort of service, which could be a Web service or an automated application.

The following table displays the set of attributes of a Service Task (where the TaskType attribute is set to "Service"), which extends the set of Task attributes (see Table 9.25):

Table 9.26 - Service Task Attributes

Attributes	Description
InMessageRef : Message	A Message for the InMessageRef attribute MUST be entered. This indicates that the Message will be received at the start of the Task, after the availability of any defined InputSets. One or more corresponding incoming Message Flows MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all incoming Message Flow, but can arrive for only one of the incoming Message Flow for a single instance of the Task.
OutMessageRef : Message	A Message for the OutMessageRef attribute MUST be entered. The sending of this message marks the completion of the Task, which may cause the production of an OutputSet. One or more corresponding outgoing Message Flow MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all outgoing Message Flow and the Message will be sent down all outgoing Message Flow at the completion of a single instance of the Task.
Implementation (Web Service | Other | Unspecified) Web Service : String	This attribute specifies the technology that will be used to send and receive the messages. A Web service is the default technology.

Receive Task

A Receive Task is a simple Task that is designed to wait for a message to arrive from an external participant (relative to the Business Process). Once the message has been received, the Task is completed.

A Receive Task is often used to start a Process. In a sense, the Process is bootstrapped by the receipt of the message. In order for the Task to Instantiate the Process it must meet one of the following conditions:

◆ The Process does not have a Start Event and the Receive Task has no incoming Sequence Flow.

◆ The Incoming Sequence Flow for the Receive Task has a source of a Start Event.

◆ Note that no other incoming Sequence Flow are allowed for the Receive Task (in particular, a loop connection from a downstream object).

The following table displays the set of attributes of a Receive Task (where the TaskType attribute is set to "Receive"), which extends the set of Task attributes (see Table 9.25):

Table 9.27 - Receive Task Attributes

Attributes	Description
MessageRef: Message	A Message for the MessageRef attribute MUST be entered. This indicates that the Message will be received by the Task. The Message in this context is equivalent to an *in-only* message pattern (Web service). One or more corresponding incoming Message Flows MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all incoming Message Flow, but can arrive for only one of the incoming Message Flow for a single instance of the Task.
Instantiate False : Boolean	Receive Tasks can be defined as the instantiation mechanism for the Process with the Instantiate attribute. This attribute MAY be set to true if the Task is the first activity after the Start Event or a starting Task if there is no Start Event (i.e., there are no incoming Sequence Flow). Multiple Tasks MAY have this attribute set to True.
Implementation (Web Service | Other | Unspecified) Web Service : String	This attribute specifies the technology that will be used to receive the message. A Web service is the default technology.

Send Task

A Send Task is a simple Task that is designed to send a message to an external participant (relative to the Business Process). Once the message has been sent, the Task is completed.

The following table displays the set of attributes of a Send Task (where the TaskType attribute is set to "Send"), which extends the set of Task attributes (see Table 9.25):

Table 9.28 - Send Task Attributes

Attributes	Description
MessageRef : Message	A Message for the MessageRef attribute MUST be entered. This indicates that the Message will be sent by the Task. The Message in this context is equivalent to an *out-only* message pattern (Web service). One or more corresponding outgoing Message Flow MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all outgoing Message Flow and the Message will be sent down all outgoing Message Flow at the completion of a single instance of the Task.
Implementation (Web Service | Other | Unspecified) Web Service : String	This attribute specifies the technology that will be used to send the message. A Web service is the default technology.

User Task

A User Task is a typical "workflow" task where a human performer performs the Task with the assistance of a software application and is scheduled through a task list manager of some sort.

The following table displays the set of attributes of a User Task (where the TaskType attribute is set to "User"), which extends the set of Task attributes (see Table 9.25):

Table 9.29 - User Task Attributes

Attributes	Description
InMessageRef : Message	A Message for the InMessageRef attribute MUST be entered. This indicates that the Message will be received at the start of the Task, after the availability of any defined InputSets. One or more corresponding incoming Message Flows MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all incoming Message Flow, but can arrive for only one of the incoming Message Flow for a single instance of the Task.
OutMessage : Message	A Message for the InMessageRef attribute MUST be entered. The sending of this message marks the completion of the Task, which may cause the production of an OutputSet. One or more corresponding outgoing Message Flow MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all outgoing Message Flow and the Message will be sent down all outgoing Message Flow at the completion of a single instance of the Task.
Implementation (Web Service | Other | Unspecified) Other : String	This attribute specifies the technology that will be used by the Performers to perform the Task.

Script Task

A Script Task is executed by a business process engine. The modeler or implementer defines a script in a language that the engine can interpret. When the Task is ready to start, the engine will execute the script. When the script is completed, the Task will also be completed.

The following table displays the set of attributes of a Script Task (where the TaskType attribute is set to "Script"), which extends the set of Task attributes (see Table 9.25):

Table 9.30 - Script Task Attributes

Attributes	Description
Script (0-1) : String	The modeler MAY include a script that can be run when the Task is performed. If a script is not included, then the Task will act equivalent to a TaskType of None.

Manual Task

A Manual Task is a Task that is expected to be performed without the aid of any business process execution engine or any application. An example of this could be a telephone technician installing a telephone at a customer location.

Reference Task

There may be times where a modeler may want to reference another activity that has been defined. If the two (or more) activities share the exact same behavior, then by one referencing the other, the attributes that define the behavior only have to be created once and maintained in only one location.

The following table displays the set of attributes of a Reference Task (where the TaskType attribute is set to "Reference"), which extends the set of Task attributes (see Table 9.25):

Table 9.31 - Reference Task Attributes

Attributes	Description
TaskRef : Task	The Task being referenced MUST be identified. The attributes for the Task element can be found in Table 9.25.

Sequence Flow Connections

Section 8.4.1, "Sequence Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Sequence Flow.

- A Task MAY be a target for Sequence Flow; it can have multiple incoming Flow. Incoming Flow MAY be from an alternative path and/or a parallel paths.

Note – If the Task has multiple incoming Sequence Flow, then this is considered uncontrolled flow. This means that when a Token arrives from one of the Paths, the Task will be instantiated. It will not wait for the arrival of Tokens from the other paths. If another Token arrives from the same path or another path, then a separate instance of the Task will be created. If the flow needs to be controlled, then the flow should converge with a Gateway that precedes the Task (see Section 9.5, "Gateways," on page 71 for more information on Gateways).

- If the Task does not have an incoming Sequence Flow, and there is no Start Event for the Process, then the Task MUST be instantiated when the process is instantiated.
    - Exceptions to this are Tasks that are defined as being Compensation activities (have the Compensation Marker). Compensation Tasks are not considered a part of the Normal Flow and MUST NOT be instantiated when the Process is instantiated.
- A Task MAY be a source for Sequence Flow; it can have multiple outgoing Flow. If there are multiple outgoing Sequence Flow, then this means that a separate parallel path is being created for each Flow.

Tokens will be generated for each outgoing Sequence Flow from the Task. The TokenIds for each of the Tokens will be set such that it can be identified that the Tokens are all from the same parallel Fork as well as the number of Tokens that exist in parallel.

- If the Task does not have an outgoing Sequence Flow, and there is no End Event for the Process, then the Task marks the end of one or more paths in the Process. When the Task ends and there are no other parallel paths active, then the Process MUST be completed.
    - Exceptions to this are Tasks that are defined as being Compensation activities (have the Compensation Marker). Compensation Tasks are not considered a part of the Normal Flow and MUST NOT mark the end of the Process.

Message Flow Connections

Section 8.4.2, "Message Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Message Flow.

Note – All Message Flow must connect two separate Pools. They can connect to the Pool boundary or to Flow Objects within the Pool boundary. They cannot connect two objects within the same Pool.

- A Task MAY be the target for Message Flow; it can have zero or more incoming Message Flow. If there multiple

incoming Message Flow, then a single Message will be applied to all the Message Flow. However, only one Message can be received, from a single Message Flow, for a given instance of the Task.

◆ A Task MAY be a source for Message Flow; it can have zero or more outgoing Message Flow. If there multiple outgoing Message Flow, then a single Message will be applied to all the Message Flow. That Message will be sent down all the outgoing Message Flow.

9.5   Gateways

Gateways are modeling elements that are used to control how Sequence Flow interact as they converge and diverge within a Process. If the flow does not need to be controlled, then a Gateway is not needed. The term "Gateway" implies that there is a gating mechanism that either allows or disallows passage through the Gateway--that is, as Tokens arrive at a Gateway, they can be merged together on input and/or split apart on output as the Gateway mechanisms are invoked. To be more descriptive, a Gateway is actually a collection of "Gates." Although the Gates are not graphically depicted, the Gates are used by the Sequence Flow of to connect to or from the Gateway.

There are different types of Gateways (as described below) and the behavior of each type Gateway will determine how many of the Gates will be available for the continuation of flow. There will be one Gate for each outgoing Sequence Flow of the Gateway.

A Gateway is a diamond (see Figure 9.16), which has been used in many flow chart notations for exclusive branching and is familiar to most modelers.

◆ A Gateway is a diamond that MUST be drawn with a single thin black line.

   ◆ The use of text, color, size, and lines for a Gateway MUST follow the rules defined in Section 8.3, "Use of Text, Color, Size, and Lines in a Diagram," on page 29.
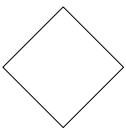
Figure 9.16 - A Gateway

Note – Although the shape of a Gateway is a diamond, it is not a requirement that incoming and outgoing Sequence Flow must connect to the corners of the diamond. Sequence Flow can connect to any position on the boundary of the Gateway shape.

Gateways can define all the types of business process Sequence Flow behavior: Decisions/branching (exclusive, inclusive, and complex), merging, forking, and joining. Thus, while the diamond has been used traditionally for exclusive decisions, BPMN extends the behavior of the diamonds to reflect any type of Sequence Flow control. Each type of Gateway will have an internal indicator or marker to show the type of Gateway that is being used (see Figure 9.17).
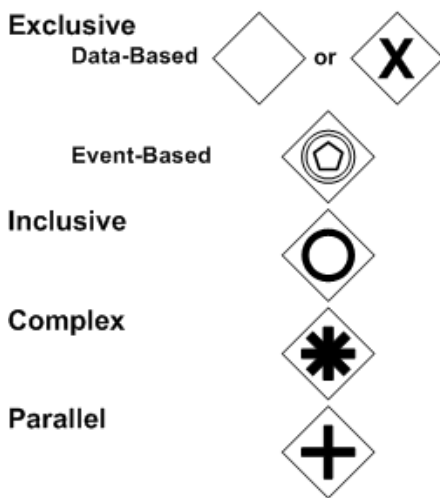
Figure 9.17 - The Different types of Gateways

◆ The internal marker associated with the Gateway MUST be placed inside the shape, in any size or location, depending on the preference of the modeler or modeling tool vendor, with the exception that the marker for the Data-Based Exclusive Gateway is not required.

The Gateways will control the flow of both diverging and/or converging Sequence Flow. That is, a particular Gateway could have multiple input Gates and multiple output Gates at the same time (there is one Sequence Flow per Gate). The type of Gateway will determine the same type of behavior for both the diverging and converging Sequence Flow. Modelers and modeling tools may want to enforce a best practice of a Gateway only performing one of these functions. Thus, it would take two sequential Gateways to first converge and then diverge the Sequence Flow.

9.5.1 Common Gateway Features

Common Gateway Attributes

The following table displays the attributes common to Gateways, and which extends the set of common Flow Object attributes (see Table 9.2):

Table 9.32 - Common Gateway Attributes

Attributes	Description
GatewayType (Exclusive | Inclusive | Complex | Parallel) Exclusive : String	GatewayType is by default Exclusive. The GatewayType MAY be set to Inclusive, Complex, or Parallel. The GatewayType will determine the behavior of the Gateway, both for incoming and outgoing Sequence Flow, and will determine the internal indicator (as shown in Figure 9.17).

Table 9.32 - Common Gateway Attributes

Attributes	Description
Gates (0-n) : Gate	There MAY be zero or more Gates (except where noted below). Zero Gates are allowed if the Gateway is last object in a Process flow and there are no Start or End Events for the Process. If there are zero or only one incoming Sequence Flow, then there MUST be at least two Gates. *For Exclusive Data-Based Gateways* When two Gates are required, one of them MAY be the DefaultGate. *For Exclusive Event-Based Gateways* There MUST be two or more Gates. (Note that this type of Gateway does not act *only* as a Merge--it is always a Decision, at least.) *For Inclusive Gateways* When two Gates are required, one of them MAY be the DefaultGate.

Common Gateway Sequence Flow Connections

This section applies to all Gateways. Additional Sequence Flow Connection rules will be specified for each type of Gateway in the sections below. See Section 8.4.1, "Sequence Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Sequence Flow.

◆ A Gateway MAY be a target for Sequence Flow; it can have zero or more incoming Sequence Flow. An incoming Flow MAY be from an alternative path or a parallel path.

   ◆ If the Gateway does not have an incoming Sequence Flow, and there is no Start Event for the Process, then the Gateway's divergence behavior, depending on the GatewayType attribute (see below), SHALL be performed when the Process is instantiated.

◆ A Gateway MAY be a source of Sequence Flow; it can have zero or more outgoing Flow.

◆ A Gateway MAY have both multiple incoming and outgoing Sequence Flow.

Note – The incoming and outgoing Sequence Flow are not required to attach to the corners of the Gateway's diamond shape. Sequence Flow can attach to any location on the boundary of a Gateway.

Message Flow Connections

This section applies to all Gateways. See Section 8.4.2, "Message Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Message Flow.

◆ An Gateway MUST NOT be a target for Message Flow.

◆ An Gateway MUST NOT be a source for Message Flow.

Gates

The following table displays the attributes of Gates, and which extends the set of common BPMN element attributes (see Table 9.1):

Table 9.33 - Gate Attributes

Attributes	Description
OutgoingSequenceFlow : SequenceFlow	Each Gate MUST have an associated (outgoing) Sequence Flow. The attributes of a Sequence Flow can be found in the Section 10.1.2, "Sequence Flow," on page 101. *For Exclusive Event-Based, Complex, and Parallel Gateways:* The Sequence Flow MUST have its Condition attribute set to None (there is not an evaluation of a condition expression). *For Exclusive Data-Based, and Inclusive Gateways:* The Sequence Flow MUST have its Condition attribute set to Expression and MUST have a valid ConditionExpression. The ConditionExpression MUST be unique for all the Gates within the Gateway. If there is only one Gate (i.e., the Gateway is acting only as a Merge), then Sequence Flow MUST have its Condition set to None. *For DefaultGates:* The Sequence Flow MUST have its Condition attribute set to Otherwise
Assignments (0-n) : Assignment	One or more assignment expressions MAY be made for each Gate. The Assignment SHALL be performed when the Gate is selected. The details of Assignment are defined in the Section B.11.3, "Assignment," on page 277.

9.5.2 Exclusive Gateways

Exclusive Gateways (Decisions) are locations within a business process where the Sequence Flow can take two or more alternative paths. This is basically the "fork in the road" for a process. For a given performance (or instance) of the process, only one of the paths can be taken (this should not be confused with forking of paths—refer to "Forking Flow" on page 111). A Decision is not an activity from the business process perspective, but is a type of Gateway that controls the Sequence Flow between activities. It can be thought of as a question that is asked at that point in the Process. The question has a defined set of alternative answers (Gates). Each Decision Gate is associated with a condition expression found within an outgoing Sequence Flow. When a Gate is chosen during the performance of the Process, the corresponding Sequence Flow is then chosen. A Token arriving at the Decision would be directed down the appropriate path, based on the chosen Gate.

The Exclusive Decision has two or more outgoing Sequence Flow, but only one of them may be taken during the performance of the Process. Thus, the Exclusive Decision defines a set of alternative paths for the Token to take as it traverses the Flow. There are two types of Exclusive Decisions: Data-Based and Event-Based.

Data-Based

The Data-Based Exclusive Gateways are the most commonly used type of Gateways. The set of Gates for Data-Based Exclusive Decisions is based on the boolean expression contained in the ConditionExpression attribute of the outgoing Sequence Flow of the Gateway. These expressions use the values of process data to determine which path should be taken (hence the name Data-Based).

Note – BPMN does not specify the format of the expressions used in Gateways or any other BPMN element that uses expressions.

- ◆ The Data-Based Exclusive Gateway MAY use a marker that is shaped like an "X" and is placed within the Gateway diamond (see Figure 9.19) to distinguish it from other Gateways. This marker is not required (see Figure 9.18).
  - ◆ A Diagram SHOULD be consistent in the use of the "X" internal indicator. That is, a Diagram SHOULD NOT have some Gateways with an indicator and some Gateways without an indicator.
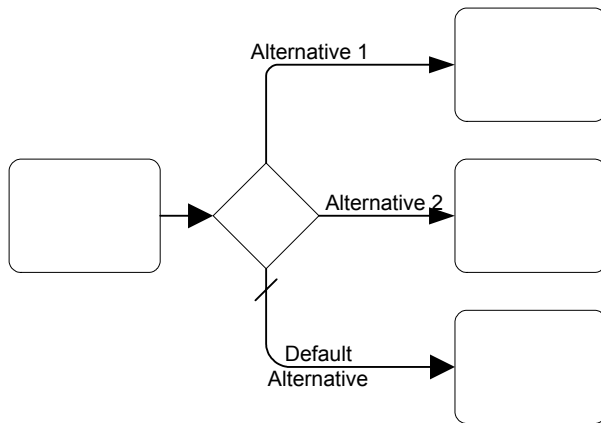
Figure 9.18 - An Exclusive Data-Based Decision (Gateway) Example without the Internal Indicator
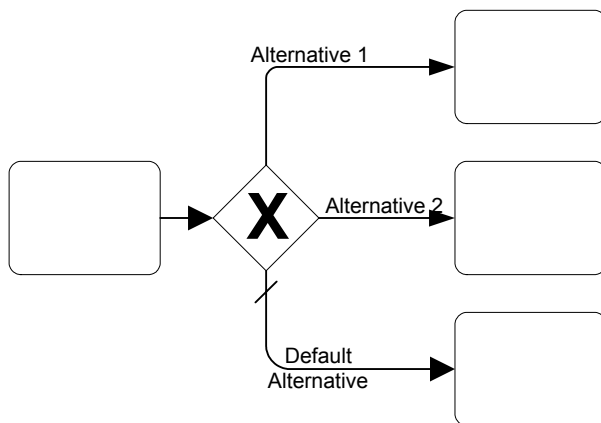
Figure 9.19 - A Data-Based Exclusive Decision (Gateway) Example with the Internal Indicator

Note – The "X" internal indicator for the Data-Based Exclusive Gateway was included in BPMN to complete the set of indicators for the different types of Gateways (see Figure 9.17). However, it is also understood that most modelers would be familiar with an empty decision diamond that represents an exclusive branching of the process and that most decisions would probably take this form. Thus, Data-Based Exclusive Gateway internal indicator was made optional so that modelers and modeling tools could create diagrams that would conform with the basic flow expectations of modelers.

The conditions for the alternative Gates should be evaluated in a specific order. The first one that evaluates as TRUE will determine the Sequence Flow that will be taken. Since the behavior of this Gateway is exclusive, any other conditions that may actually be TRUE will be ignored--only one Gate can be chosen. One of the Gates may be "default" (or otherwise), and is the last Gate considered. This means that if none of the other Gates are chosen, then the default Gate will be chosen—along with its associated Sequence Flow.

The default Gate is not mandatory for a Gateway. This means that if it is not used, then it is up to the modeler to insure that at least one Gate be valid at runtime. BPMN does not specify what will happen if there are no valid Gates. However, BPMN does specify that there MUST NOT be implicit flow and that all Normal Flow of a Process must be expressed through Sequence Flow. This would mean that a Process Model that has a Gateway that potentially does not have a valid Gate at runtime is an invalid model.
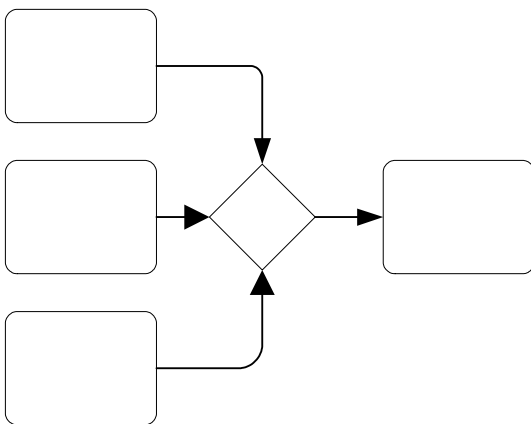
Figure 9.20 - An Exclusive Merge (Gateway) (without the Internal Indicator)

Exclusive Gateways can also be used as a merge (see Figure 9.20) for alternative Sequence Flow, although it is rarely required for the modeler to use them this way. The merging behavior of the Gateway can also be modeled as seen in Figure 9.21. The behavior of Figure 9.20 and Figure 9.21 are the same if all the incoming flow are alternative.
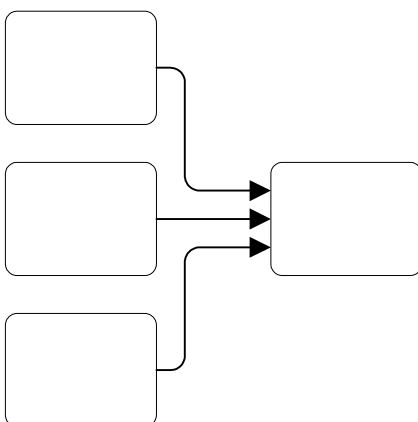
Figure 9.21 - Uncontrolled Merging of Sequence Flow

There are certain situations where an Exclusive Gateway is required to act as a merging object. In Figure 9.23 an Exclusive Gateway (labeled "Merge") merges two alternative Sequence Flow that were generated by an upstream Decision. The alternative Sequence Flow are merged in preparation for an Parallel Gateway that synchronizes a set of parallel Sequence Flow that were generated even further upstream. If the merging Gateway was not used, then there would have been four incoming Sequence Flow into the Parallel Gateway. However, only three of the four Sequence Flow would ever pass a Token at one time. Thus, the Gateway would be waiting for a fourth Token that would never arrive. Thus, the Process would be stuck at the point of the Parallel Gateway.
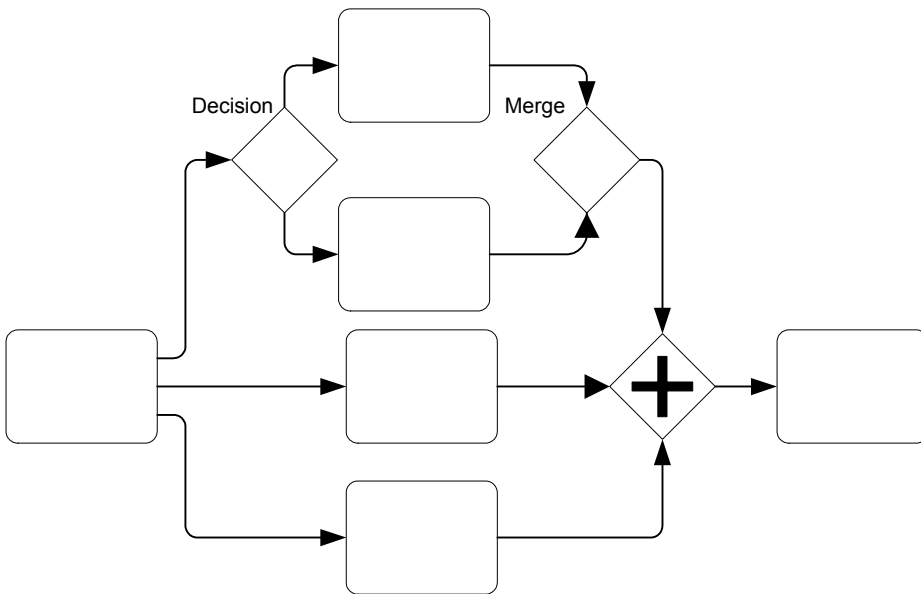
Figure 9.22 - Exclusive Gateway that merges Sequence Flow prior to an Parallel Gateway

In simple situations, Exclusive Gateways need not be used for merging Sequence Flow, but there are more complex situations where they are required. Thus, a modeler should always be aware of the behavior of a situation where Sequence Flow are uncontrolled. Some modelers or modeling tools may, in fact, require that Exclusive Gateways be used in all situations as a matter of Best Practice.

Attributes

The following table displays the attributes for an Data-Based Exclusive Gateway. These attributes only apply if the GatewayType attribute is set to Exclusive. The following attributes extend the set of common Gateway attributes (see Table 9.32):

Table 9.34 - Data-Based Exclusive Gateway Attributes

Attributes	Description
ExclusiveType (Data | Event) Data : String	ExclusiveType is by default Data. The ExclusiveType MAY be set to Event. Since Data-Based Exclusive Gateways are the subject of this section, the attribute MUST be set to Data for the attributes and behavior defined in this section to apply to the Gateway.

Table 9.34 - Data-Based Exclusive Gateway Attributes

Attributes	Description
MarkerVisible False : Boolean	This attribute determines if the Exclusive Marker is displayed in the center of the Gateway diamond (an "X"). The marker is displayed if the attribute is True and it is not displayed if the attribute is False. By default, the marker is not displayed.
DefaultGate (0-1) : Gate	A Default Gate MAY be specified (see Section , "Gates," on page 74).

Sequence Flow Connections

This section extends the basic Gateway Sequence Flow connection rules as defined in "Common Gateway Sequence Flow Connections" on page 73. Section 8.4.1, "Sequence Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Sequence Flow.

To define the exclusive nature of this Gateway's behavior for converging Sequence Flow:

◆ If there are multiple incoming Sequence Flow, all of them will be used to continue the flow of the Process (as if there were no Gateway). That is,

    ◆ Process flow SHALL continue when a signal (a Token) arrives from any of a set of Sequence Flow.

        ◆ Signals from other Sequence Flow within that set may arrive at other times and the flow will continue when they arrive as well, without consideration or synchronization of signals that have arrived from other Sequence Flow.

To define the exclusive nature of this Gateway's behavior for diverging Sequence Flow:

◆ If there are multiple outgoing Sequence Flow, then only one Gate (or the DefaultGate) SHALL be selected during performance of the Process.

    ◆ The Gate SHALL be chosen based on the result of evaluating the ConditionExpression that is defined for the Sequence Flow associated with the Gate.

        ◆ The Conditions associated with the Gates SHALL be evaluated in the order in which the Gates appear on the list for the Gateway.

        ◆ If a ConditionExpression is evaluated as "TRUE," then that Gate SHALL be chosen and any Gates remaining on the list MUST NOT be evaluated.

        ◆ If none of the ConditionExpressions for the Gates are evaluated as "TRUE," then the DefaultGate SHALL be chosen.

Note – If the Gateway does not have a DefaultGate and none of the Gate ConditionExpressions are evaluated as "TRUE," then the Process is considered to have an invalid model.

Event-Based

The inclusion of Event-Based Exclusive Gateways is the result of recent developments in the handling of distributed systems (e.g., with pi-calculus) and was derived from the BPEL4WS *pick*. On the input side, their behavior is the same as a Data-Based Exclusive Gateway (see "Data-Based" on page 74). On the output side, the basic idea is that this Decision represents a branching point in the process where the alternatives are based on events that occurs at that point in the Process, rather than the evaluation of expressions using process data. A specific event, usually the receipt of a message, determines which of the paths will be taken. For example, if a company is waiting for a response from a customer, they

will perform one set of activities if the customer responds "Yes" and another set of activities if the customer responds "No." The customer's response determines which path is taken. The identity of the Message determines which path is taken. That is, the "Yes" Message and the "No" message are different messages—they are not the same message with different values within a property of the Message. The receipt of the message can be modeled with a Task of TaskType Receive or an Intermediate Event with a Message Trigger. In addition to Messages, other Triggers for Intermediate Events can be used, such as Timers.

◆ The Event-Based Exclusive Gateway MUST use a marker that is the same as the Multiple Intermediate Event and is placed within the Gateway diamond (see Figure 9.23 and Figure 9.24) to distinguish it from other Gateways.

◆ The Event-Based Exclusive Decisions are configured by having outgoing Sequence Flow target a Task of TaskType Receive or an Intermediate Event (see Figure 9.23 and Figure 9.24).

   ◆ All of the outgoing Sequence Flow must have this type of target; there cannot be a mixing of condition expressions and Intermediate Events for a given Decision.
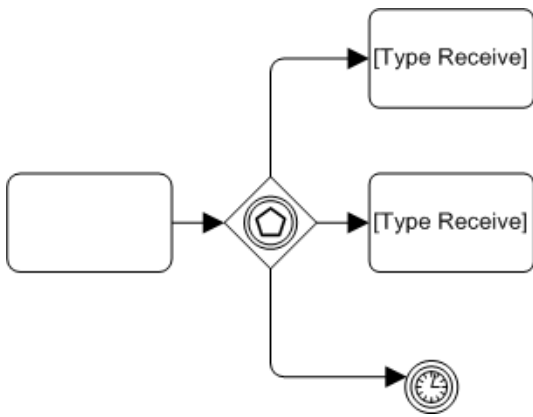
Figure 9.23 - An Event-Based Decision (Gateway) Example Using Receive Tasks
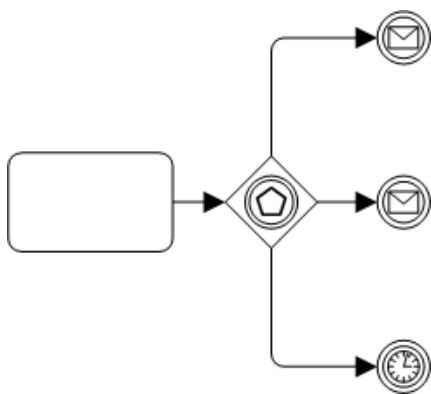
Figure 9.24 - An Event-Based Decision (Gateway) Example Using Message Events

Because this Gateway is an Exclusive Gateway, the merging functionality for the Event-Based Exclusive Gateway is the same as the Data-Based Exclusive Gateway described in the previous section.

A Gateway can be used to start a Process. In a sense, the Process is bootstrapped by the receipt of a message. The receipt of any of the messages defined by the Gateway configuration will instantiate the Process. Thus, the Gateway provides a set of alternative ways for the Process to begin.

In order for the Gateway to Instantiate the Process it must meet one of the following conditions:

- The Process does not have a Start Event and the Gateway has no incoming Sequence Flow.
- The Incoming Sequence Flow for the Gateway has a source of a Start Event.
  - Note that no other incoming Sequence Flow are allowed for the Gateway (in particular, a loop connection from a downstream object).
- The Targets for the Gateway's outgoing Sequence Flow MUST NOT be a Timer Intermediate Event.

Attributes

The following table displays the attributes for an Event-Based Exclusive Gateway. These attributes only apply if the GatewayType attribute is set to Exclusive. The following attributes extend the set of common Gateway attributes (see Table 9.38):S

Table 9.35 - Event-Based Exclusive Gateway Attributes

Attributes	Description
ExclusiveType (Data | Event) Event : String	ExclusiveType is by default Data. The ExclusiveType MAY be set to Event. Since Event-Based Exclusive Gateways is the subject of this section, the attribute MUST be set to Event for the attributes and behavior defined in this section to apply to the Gateway.
Instantiate False : Boolean	Event-Based Gateways can be defined as the instantiation mechanism for the Process with the Instantiate attribute. This attribute MAY be set to true if the Gateway is the first element after the Start Event or a starting Gateway if there is no Start Event (i.e., there are no incoming Sequence Flow).

Sequence Flow Connections

This section extends the basic Gateway Sequence Flow connection rules as defined in "Common Gateway Sequence Flow Connections" on page 73. See Section 8.4.1, "Sequence Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Sequence Flow.

To define the exclusive nature of this Gateway's behavior for converging Sequence Flow:

- If there are multiple incoming Sequence Flow, all of them will be used to continue the flow of the Process (as if there were no Gateway). That is,
  - Process flow SHALL continue when a signal (a Token) arrives from any of a set of Sequence Flow.
    - Signals from other Sequence Flow within that set may arrive at other times and the flow will continue when they arrive as well, without consideration or synchronization of signals that have arrived from other Sequence Flow.

To define the exclusive nature of this Gateway's behavior for diverging Sequence Flow:

- Only one Gate SHALL be selected during performance of the Process.
  - The Gate SHALL be chosen based on the Target of the Gate's Sequence Flow.
    - If a Target is instantiated (e.g., a message is received or a time is exceeded), then that Gate SHALL be

chosen and the remaining Gates MUST NOT be evaluated (i.e., their Targets will be disabled).

◆ The outgoing Sequence Flow Condition attribute MUST be set to None.

◆ The Target of the Gateway's outgoing Sequence Flow MUST be one of the following objects:

  ◆ Task with the TaskType attribute set to Receive.

  ◆ Intermediate Event with the Trigger attribute set to Message, Timer, Signal, or Conditional.

    ◆ If one Gate Target is a Task, then an Intermediate Event with a Trigger Message MUST NOT be used as a Target for another Gate. That is, messages MUST be received by only Receive Tasks or only Message Events, but not a mixture of both for a given Gateway.

9.5.3 Inclusive Gateways

This Decision represents a branching point where Alternatives are based on conditional expressions contained within outgoing Sequence Flow. However, in this case, the True evaluation of one condition expression does not exclude the evaluation of other condition expressions. All Sequence Flow with a True evaluation will be traversed by a Token. In some sense it like is a grouping of related independent Binary (Yes/No) Decisions--and can be modeled that way. 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.

Note – If none of the Inclusive Decision Gate ConditionExpressions are evaluated as "TRUE," then the Process is considered to have an invalid model.

There are two mechanism for modeling this type of Decision:

The first method for modeling Inclusive Decision situations does not actually use an Inclusive Gateway, but instead uses a collection of conditional Sequence Flow, marked with mini-diamonds--the Gates without the Gateway (see Figure 9.25). Conditional Sequence Flow have their Condition attribute set to Expression and the ConditionExpression attribute set to a boolean mathematical expression based on information available to the Process. These Sequence Flow are indicated by a "mini-diamond" marker at the start of the Sequence Flow line.
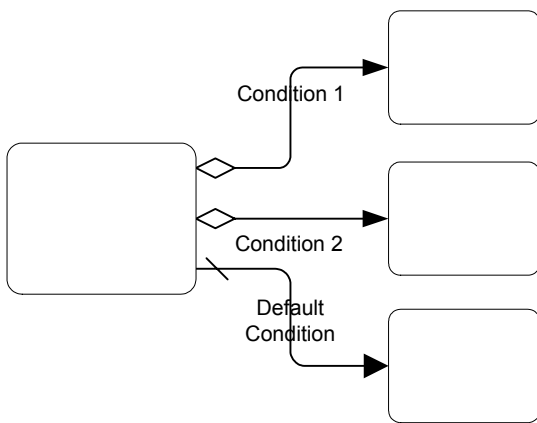
Figure 9.25 - An Inclusive Decision using Conditional Sequence Flow

There are some restrictions in using the conditional Sequence Flow (with mini-diamonds):

- The source object MUST NOT be an Event. The source object MAY a Gateway, but the mini-diamond MUST NOT be displayed in this case. The source object MAY be an activity (Task or Sub-Process) and the mini-diamond SHALL be displayed in this case.
    - A source Gateway MUST NOT be of type Parallel.
- If a conditional Sequence Flow is used from a source activity, then there MUST be at least one other outgoing Sequence Flow from that activity
    - The additional Sequence Flow(s) MAY also be conditional, but it is not required that they are conditional.

The second method for modeling Inclusive Decision situations uses an Inclusive Gateway (see Figure 9.26), sometimes in combination with other Gateways. A marker will be placed in the center of the Gateway to indicate that the behavior of the Gateway is inclusive.

- ◆ The Inclusive Gateway MUST use a marker that is in the shape of a circle or an "O" and is placed within the Gateway diamond (see Figure 9.26) to distinguish it from other Gateways.
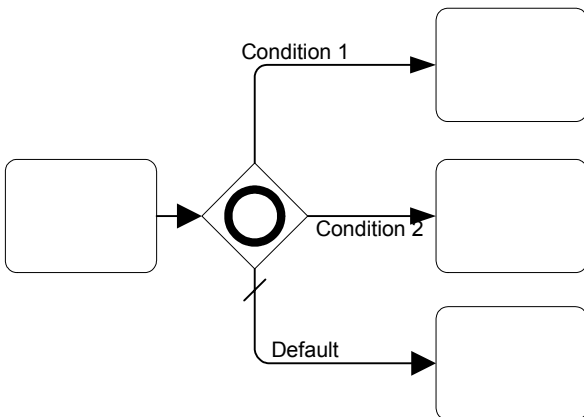
Figure 9.26 - An Inclusive Decision using an Inclusive Gateway

The behavior of the model depicted in Figure 9.25 is equivalent to the behavior of the model depicted in Figure 9.26. Again, it is up to the modeler to insure that at least one of the conditions will be TRUE when the Process is performed.

When the Inclusive Gateway is used as a Merge, it will synchronize all Tokens that have been produced upstream, but at most one for each incoming Sequence Flow (refer to the section entitled "XXXX" for a definition of Token flow semantics). Note: Tokens with a loop are upstream of every node in the loop. It requires that Tokens for all Sequence Flow that were actually produced by an upstream (by an Inclusive situation, for example) be synchronized. If an upstream Inclusive produces two out of a possible three Tokens, then a downstream Inclusive will synchronize those two Tokens and not wait for another Token, even though there are three incoming Sequence Flow (see Figure 9.27).
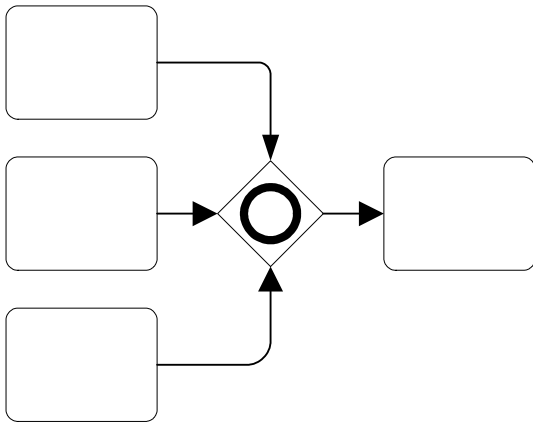
Figure 9.27 - An Inclusive Gateway Merging Sequence Flow

Attributes

The following table displays the attributes for an Inclusive Gateway. These attributes only apply if the GatewayType attribute is set to Inclusive. The following attributes extend the set of common Gateway attributes (see Table 9.32):

Table 9.36 - Inclusive Gateway Attributes

Attributes	Description
DefaultGate (0-1) : Gate	A Default Gate MAY be specified (see Section , "Gates," on page 74).

Sequence Flow Connections

This section extends the basic Gateway Sequence Flow connection rules as defined in "Common Gateway Sequence Flow Connections" on page 73.See Section 8.4.1, "Sequence Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Sequence Flow.

To define the inclusive nature of this Gateway's behavior for converging Sequence Flow:

◆ If there are multiple incoming Sequence Flow, one or more of them will be used to continue the flow of the Process. That is,

    ◆ Process flow SHALL continue when the signals (Tokens) arrive from all of the incoming Sequence Flow that are expecting a signal based on the upstream structure of the Process (e.g., an upstream Inclusive Decision).

        ◆ Some of the incoming Sequence Flow will not have signals and the pattern of which Sequence Flow will have signals may change for different instantiations of the Process.

Note – Incoming Sequence Flow that have a source that is a downstream activity (that is, is part of a loop) will be treated differently than those that have an upstream source. They will be considered as part of a different set of Sequence Flow from those Sequence Flow that have a source that is an upstream activity.

To define the inclusive nature of this Gateway's behavior for diverging Sequence Flow:

◆ One or more Gates SHALL be selected during performance of the Process.

◆ The Gates SHALL be chosen based on the Condition expression that is defined for the Sequence Flow associated with the Gates.

    ◆ The Condition associated with all Gates SHALL be evaluated.

    ◆ If a Condition is evaluated as "TRUE," then that Gate SHALL be chosen, independent of what other Gates have or have not been chosen.

    ◆ If none of the ConditionExpressions for the Gates are evaluated as "TRUE," then the DefaultGate SHALL be chosen.

9.5.4 Complex Gateways

BPMN includes a Complex Gateway to handle situations that are not easily handled through the other types of Gateways. Complex Gateways can also be used to combine a set of linked simple Gateways into a single, more compact situation. Modelers can provide complex expressions that determine the merging and/or splitting behavior of the Gateway.

◆ The Complex Gateway MUST use a marker that is in the shape of an asterisk and is placed within the Gateway diamond (see Figure 9.28) to distinguish it from other Gateways.

When the Gateway is used as a Decision (see Figure 9.28), then an expression determines which of the outgoing Sequence Flow will be chosen for the Process to continue. The expression may refer to process data and the status of the incoming Sequence Flow. For example, an expression may evaluate Process data and then select different sets of outgoing Sequence Flow, based on the results of the evaluation. However, The expression should be designed so that at least one of the outgoing Sequence Flow will be chosen.
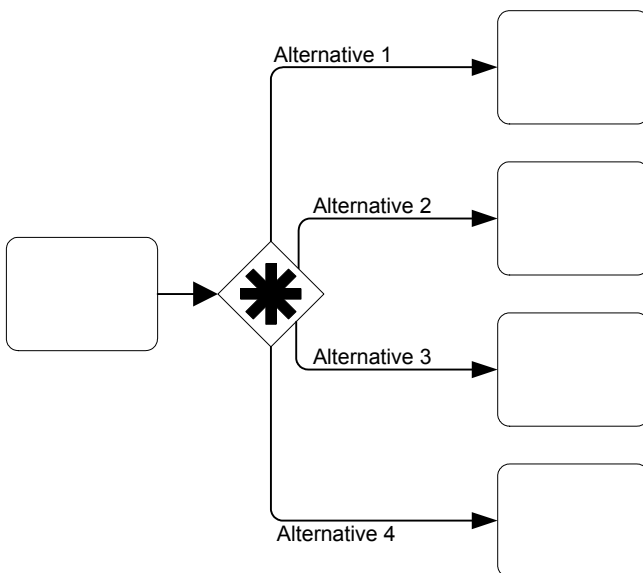
Figure 9.28 - A Complex Decision (Gateway)

When the Gateway is used as a Merge (see Figure 9.29), then there will be an expression that will determine which of the incoming Sequence Flow will be required for the Process to continue. The expression may refer to process data and the status of the incoming Sequence Flow. For example, an expression may specify that any 3 out of 5 incoming Tokens will continue the Process. Another example would be an expression that specifies that a Token is required from Sequence Flow "a" and that a Token from either Sequence Flow "b" or "c" is acceptable. However, the expression should be designed so that the Process is not stalled at that location.
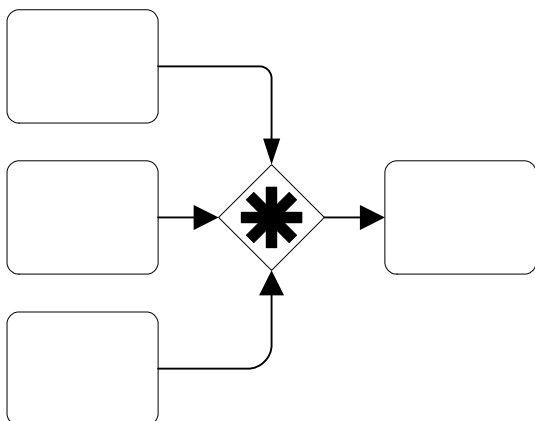
Figure 9.29 - A Complex Merge (Gateway)

Attributes

The following table displays the attributes for a Complex Gateway. These attributes only apply if the GatewayType attribute is set to Complex. The following attributes extend the set of common Gateway attributes (see Table 9.32):

Table 9.37 - Complex Gateway Attributes

Attributes	Description
IncomingCondition (0-1) : Expression	If there are multiple incoming Sequence Flow, an IncomingCondition expression MUST be set by the modeler. This will consist of an expression that can reference Sequence Flow names and/or Process Properties (Data).
OutgoingCondition (0-1) : Expression	If there are multiple outgoing Sequence Flow, an OutgoingCondition expression MUST be set by the modeler. This will consist of an expression that can reference (outgoing) Sequence Flow Ids and/or Process Properties (Data).

Sequence Flow Connections

This section extends the basic Gateway Sequence Flow connection rules as defined in "Common Gateway Sequence Flow Connections" on page 73. See Section 8.4.1, "Sequence Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Sequence Flow.

To define the complex nature of this Gateway's behavior for converging Sequence Flow:

◆ If there are multiple incoming Sequence Flow, one or more of them will be used to continue the flow of the Process. The exact combination of incoming Sequence Flow will be determined by the Gateway's IncomingCondition expression.

◆ Process flow SHALL continue when the appropriate number of signals (Tokens) arrives from appropriate incoming Sequence Flow.

◆ Signals from other Sequence Flow within that set MAY arrive, but they MUST NOT be used to continue the flow of the Process.

Note – Incoming Sequence Flow that have a source that is a downstream activity (that is, is part of a loop) will be treated differently than those that have an upstream source. They will be considered as part of a different set of Sequence Flow from those Sequence Flow that have a source that is an upstream activity.

To define the inclusive nature of this Gateway's behavior for diverging Sequence Flow:

◆ One or more Gates SHALL be selected during performance of the Process.

   ◆ The Gates SHALL be chosen based on the Gateway's OutgoingCondition expression.

9.5.5 Parallel Gateways

Parallel Gateways provide a mechanism to synchronize parallel flow and to create parallel flow. These Gateways are not required to create parallel flow, but they can be used to clarify the behavior of complex situations where a string of Gateways are used and parallel flow is required. In addition, some modelers may wish to create a "best practice" where Parallel Gateways are always used for creating parallel paths. This practice will create an extra modeling element where one is not required, but will provide a balanced approach where forking and joining elements can be paired up.

◆ The Parallel Gateway MUST use a marker that is in the shape of an plus sign and is placed within the Gateway diamond (see Figure 9.30) to distinguish it from other Gateways.
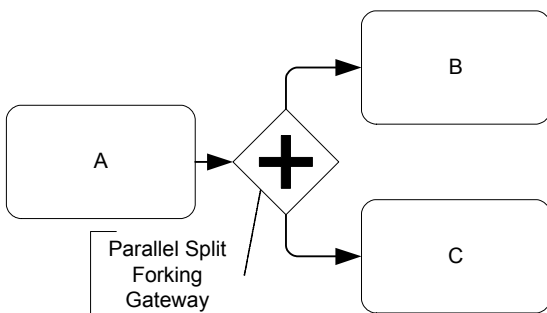
Figure 9.30 - A Parallel Gateway

Parallel Gateways are used for synchronizing parallel flow.
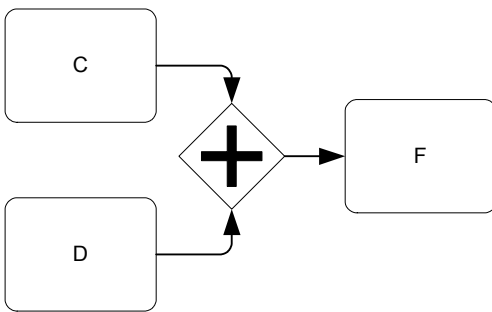
Figure 9.31 - Joining – the joining of parallel paths

Attributes

Parallel Gateways do not have any additional Attributes beyond the common Gateway Attributes (see Table 9.32).

Sequence Flow Connections

This section extends the basic Gateway Sequence Flow connection rules as defined in "Common Gateway Sequence Flow Connections" on page 73. See Section 8.4.1, "Sequence Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Sequence Flow.

To define the parallel nature of this Gateway's behavior for converging Sequence Flow:

◆ If there are multiple incoming Sequence Flow, all of them will be used to continue the flow of the Process--the flow will be synchronized. That is,

   ◆ Process flow SHALL continue when a signal (a Token) has arrived from all of a set of Sequence Flow (i.e., the process will wait for all signals to arrive before it can continue).

Note – Incoming Sequence Flow that have a source that is a downstream activity (that is, is part of a loop) will be treated differently than those that have an upstream source. They will be considered as part of a different set of Sequence Flow from those Sequence Flow that have a source that is an upstream activity.

To define the parallel nature of this Gateway's behavior for diverging Sequence Flow:

◆ All Gates SHALL be selected during performance of the Process.

9.6   Swimlanes (Pools and Lanes)

BPMN uses the concept known as "swimlanes" to help partition and/organize activities. It is possible that a BPMN Diagram may depict more than one private process, as well as the processes that show the collaboration between private processes or Participants. If so, then each private business process will be considered as being performed by different Participants. Graphically, each Participant will be partitioned; that is, will be contained within a rectangular box call a "Pool." Pools can have sub-Swimlanes that are called, simply, "Lanes."

Section 7.1.1, "Uses of BPMN," on page 12 describes the uses of BPMN for modeling private processes and the interactions of processes in B2B scenarios. Pools and Lanes are designed to support these uses of BPMN.

9.6.1 Common Swimlane Attributes

The following table displays a set of common attributes for Swimlanes (Pools and Lanes), and which extends the set of common BPMN Element attributes (see Table 9.1):

Table 9.38 - Common Swimlane Attributes

Attributes	Description
Name : String	Name is an attribute that is text description of the Swimlane.

9.6.2 Pool

A Pool represents a Participant in the Process. A Participant can be a specific business entity (e.g, a company) or can be a more general business role (e.g., a buyer, seller, or manufacturer). Graphically, a Pool is a container for partitioning a Process from other Pools, when modeling business-to-business situations, although a Pool need not have any internal details (i.e., it can be a "black box").

◆ A Pool is a square-cornered rectangle that MUST be drawn with a solid single black line (as seen in Figure 9.32).

　　◆ One, and only one, Pool in a diagram MAY be presented without a boundary. If there is more than one Pool in the diagram, then the remaining Pools MUST have a boundary.

　　◆ The use of text, color, size, and lines for a Pool MUST follow the rules defined in Section 8.3, "Use of Text, Color, Size, and Lines in a Diagram," on page 29.
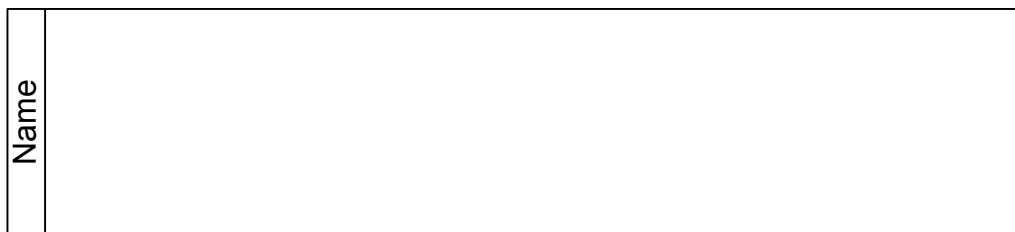
Figure 9.32 - A Pool

To help with the clarity of the Diagram, A Pool will extend the entire length of the Diagram, either horizontally or vertically. However, there is no specific restriction to the size and/or positioning of a Pool. Modelers and modeling tools can use Pools (and Lanes) in a flexible manner in the interest of conserving the "real estate" of a Diagram on a screen or a printed page.

A Pool acts as the container for the Sequence Flow between activities. The Sequence Flow can cross the boundaries between Lanes of a Pool, but cannot cross the boundaries of a Pool. The interaction between Pools, e.g., in a B2B context, is shown through Message Flow.

Another aspect of Pools is whether or not there is any activity detailed within the Pool. Thus, a given Pool may be shown as a "White Box," with all details exposed, or as a "Black Box," with all details hidden. No Sequence Flow is associated with a "Black Box" Pool, but Message Flow can attach to its boundaries (see Figure 9.33).
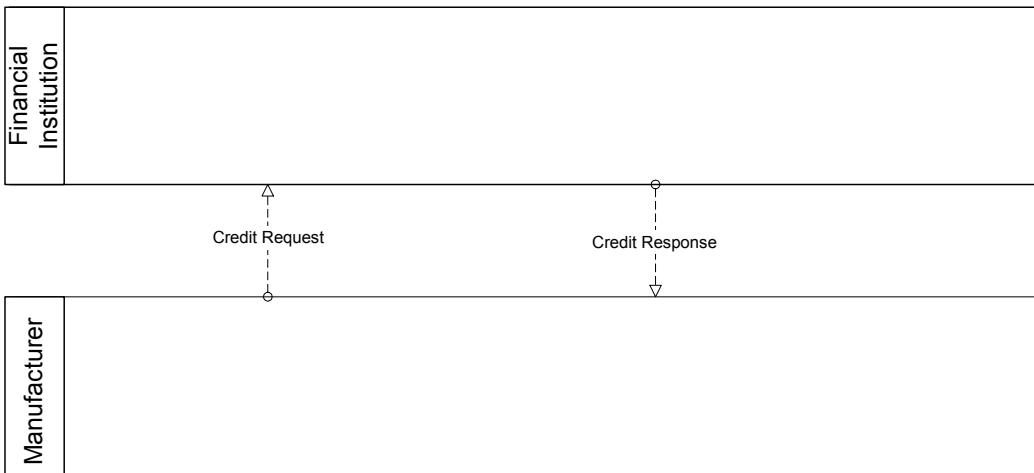
Figure 9.33 - Message Flow connecting to the boundaries of two Pools

For a "White Box" Pool, the activities within are organized by Sequence Flow. Message Flow can cross the Pool boundary to attach to the appropriate activity (see Figure 9.34).
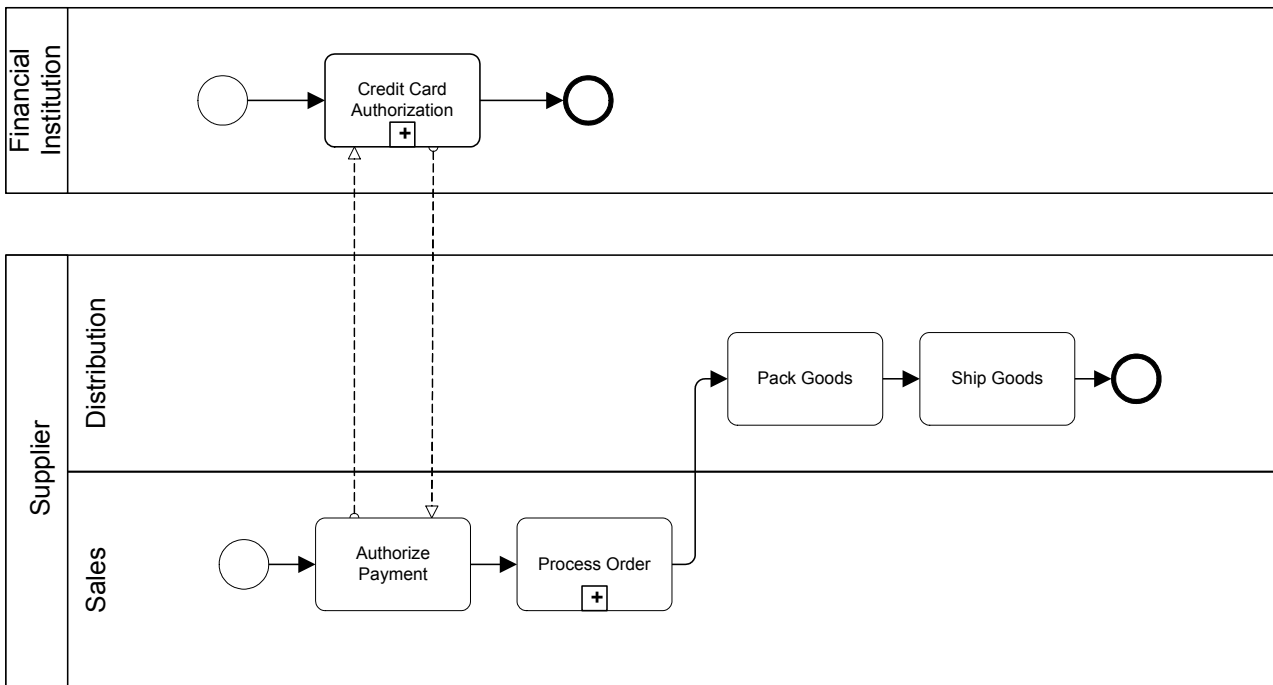
Figure 9.34 - Message Flow connecting to Flow Objects within two Pools

All BPDs contain at least one Pool. In most cases, a BPD that consists of a single Pool will only display the activities of the Process and not display the boundaries of the Pool. Furthermore, a BPD may show the "main" Pool without boundaries. In such cases there can be, at most, only one invisibly-bounded pool in the diagram and the name of that pool

SHALL be the same as the diagram. Consequently, the activities that represent the work performed from the point of view of the modeler or the modeler's organization are considered "internal" activities and need not be surrounded by the boundaries of a Pool, while the other Pools in the Diagram must have their boundary. (see Figure 9.35)
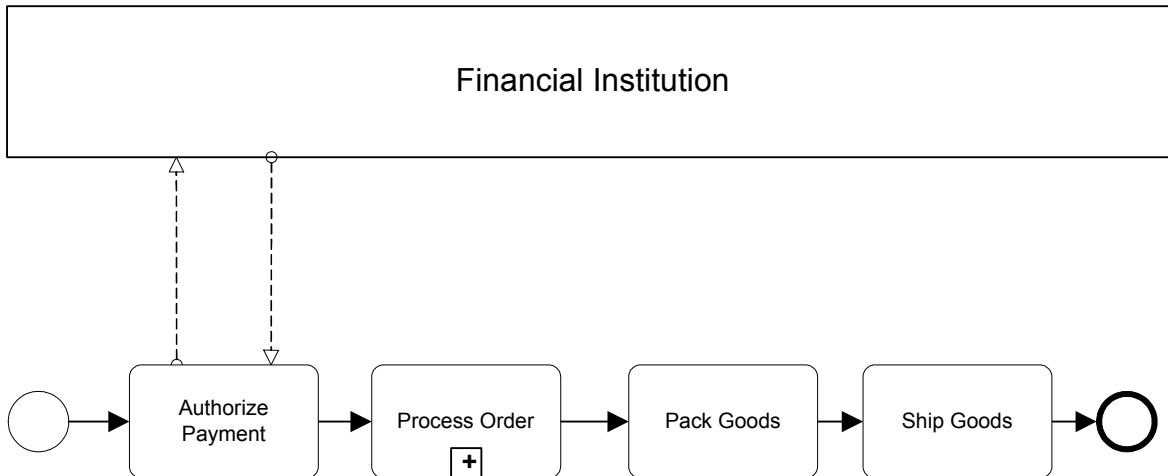
Figure 9.35 - Main (Internal) Pool without boundaries

Attributes

The following table displays the identified attributes of a Pool, and which extends the set of common Swimlane attributes (see Table 9.40):

Table 9.39 - Pool Attributes

Attributes	Description
ProcessRef (0-1) : Process	The ProcessRef attribute defines the Process that is contained within the Pool. Each Pool MAY have a Process. The attributes for a Process can be found in Section 8.6, "Processes," on page 32.
ParticipantRef : Participant	The Modeler MUST define the Participant for a Pool. The Participant can be either a Role or an Entity. This defines the role that a particular Entity or Role the Pool will play in a Diagram that includes collaboration. The attributes for a Participant can be found in Section B.11.14, "Participant," on page 284.
Lanes (1-n) : Lane	There MUST be one or more Lanes within a Pool. The attributes for a Lane can be found in Section 9.6.3, "Lane," on page 91.
BoundaryVisible True : Boolean	This attribute defines if the rectangular boundary for the Pool is visible. Only one Pool in the Diagram MAY have the attribute set to False.
MainPool False : Boolean	This attribute defines if the Pool is the "main" Pool or the focus of the diagram. Only one Pool in the Diagram MAY have the attribute set to True.

9.6.3 Lane

A Lane is a sub-partition within a Pool and will extend the entire length of the Pool, either vertically (see Figure 9.36) or horizontally (see Figure 9.37). If the pool is invisibly bounded, the lane associated with the pool must extend the entire length of the pool. Text associated with the Lane (e.g., its name and/or any attribute) can be placed inside the shape, in any direction or location, depending on the preference of the modeler or modeling tool vendor. Our examples place the name as a banner on the left side (for horizontal Pools) or at the top (for vertical Pools) on the other side of the line that separates the Pool name, however, this is not a requirement.
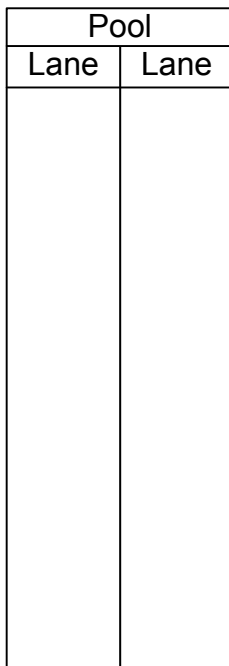
Figure 9.36 - Two Lanes in a Vertical Pool

Figure 9.37 - Two Lanes in a Horizontal Pool

Lanes are used to organize and categorize activities within a Pool. The meaning of the Lanes is up to the modeler. BPMN does not specify the usage of Lanes. Lanes are often used for such things as internal roles (e.g., Manager, Associate), systems (e.g., an enterprise application), an internal department (e.g., shipping, finance), etc. In addition, Lanes can be nested (see Figure 9.38) or defined in a matrix. For example, there could be an outer set of Lanes for company departments and then an inner set of Lanes for roles within each department.
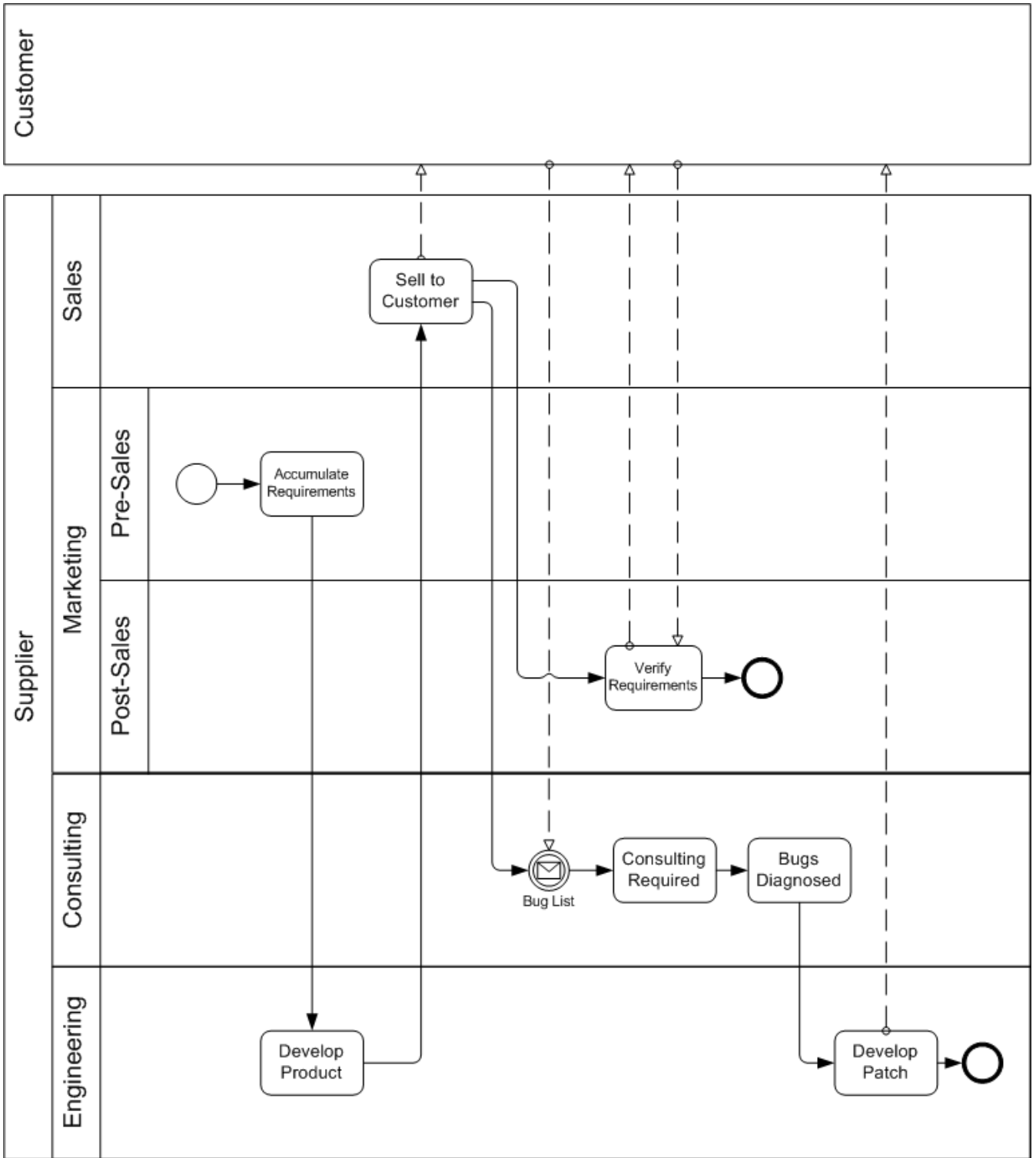
Figure 9.38 - An Example of Nested Lanes

Attributes

The following table displays the identified attributes of a Lane, and which extends the set of common Swimlane attributes (see Table 9.43):

Table 9.40 - Lane Attributes

Attributes	Description
Lanes (0-*) : Lane	This attribute identifies any Lanes that are nested within the current Lane.

9.7   Artifacts

BPMN provides modelers with the capability of showing additional information about a Process that is not directly related to the Sequence Flow or Message Flow of the Process.

At this point, BPMN provides three standard Artifacts: A Data Object, a Group, and an Annotation. Additional standard Artifacts may be added to the BPMN specification in later versions. A modeler or modeling tool may extend a BPD and add new types of Artifacts to a Diagram. Any new Artifact must follow the Sequence Flow and Message Flow connection rules (listed below). Associations can be used to link Artifacts to Flow Objects (see Section 10.1.4, "Association," on page 105).

9.7.1 Common Artifact Definitions

The following sections provide definitions that a common to all Artifacts.

Common Artifact Attributes

The following table displays the identified attributes common to Artifacts, and which extends the set of common BPMN Element attributes (see Table 9.1):

Table 9.41 - Common Artifact Attributes

Attributes	Description
ArtifactType (DataObject | Group | Annotation) DataObject : String	The ArtifactType MAY be set to DataObject, Group, or Annotation. The ArtifactType list MAY be extended to include new types.

Artifact Sequence Flow Connections

See Section 8.4.1, "Sequence Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Sequence Flow.

◆   An Artifact MUST NOT be a target for Sequence Flow.

◆   An Artifact MUST NOT be a source for Sequence Flow.

Artifact Message Flow Connections

See Section 8.4.2, "Message Flow Rules," on page 30 for the entire set of objects and how they may be source or targets of Message Flow.

- An Artifact MUST NOT be a target for Message Flow.

- An Artifact MUST NOT be a source for Message Flow.

9.7.2 Data Object

In BPMN, a Data Object is considered an Artifact and not a Flow Object. They are considered an Artifact 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. That is, how documents, data, and other objects are used and updated during the Process. While the name "Data Object" may imply an electronic document, they can be used to represent many different types of objects, both electronic and physical.

In general, BPMN will not standardize many modeling Artifacts. These will mainly be up to modelers and modeling tool vendors to create for their own purposes. However, equivalents of the BPMN Data Object are used by Document Management oriented workflow systems and many other process modeling methodologies. Thus, this object is used enough that it is important to standardize its shape and behavior.

- A Data Object is a portrait-oriented rectangle that has its upper-right corner folded over that MUST be drawn with a solid single black line (as seen in Figure 9.39).

   - The use of text, color, size, and lines for a Data Object MUST follow the rules defined in Section 8.3, "Use of Text, Color, Size, and Lines in a Diagram," on page 29.

Name
[State]

Figure 9.39 - A Data Object

As an Artifact, Data Objects generally will be associated with Flow Objects. An Association will be used to make the connection between the Data Object and the Flow Object. This means that the behavior of the Process can be modeled without Data Objects for modelers who want to reduce clutter. The same Process can be modeled with Data Objects for modelers who want to include more information without changing the basic behavior of the Process.

In some cases, the Data Object will be shown being sent from one activity to another, via a Sequence Flow (see Figure 9.40). Data Objects will also be associated with Message Flow. They are not to be confused with the message itself, but could be thought of as the "payload" or content of some messages.

Send Invoice → Make Payment

Invoice
[Approved]
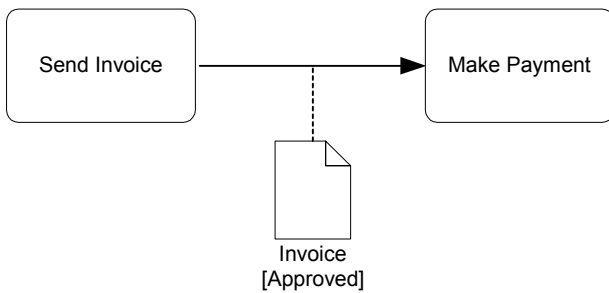
Figure 9.40 - A Data Object associated with a Sequence Flow

In other cases, the same Data Object will be shown as being an input, then an output of a Process (see Figure 9.41). Directionality added to the Association will show whether the Data Object is an input or an output. Also, the state attribute of the Data Object can change to show the impact of the Process on the Data Object.

Approve Purchase
Order

Purchase Order
[Complete]

Purchase Order
[Approved]
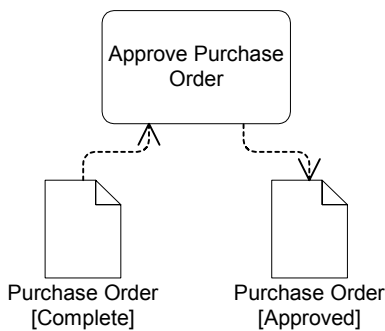
Figure 9.41 - Data Objects shown as inputs and outputs

Attributes

The following table displays the attributes for Data Objects, which extends the set of common Artifact attributes (see Table 9.41). These attributes only apply if the ArtifactType attribute is set to DataObject:

Table 9.42 - Data Object Attributes

Attributes	Description
Name : String	Name is an attribute that is text description of the object.
State (0-1) : String	State is an optional attribute that indicates the impact the Process has had on the Data Object. Multiple Data Objects with the same name MAY share the same state within one Process.
Properties (0-n) : Property	Modeler-defined Properties MAY be added to a Data Object. The fully delineated name of these properties is "<process name>.<task name>.<property name>" (e.g., "Add Customer.Credit Report.Score"). Further details about the definition of a Property can be found in Section B.11.15, "Property," on page 285.

9.7.3 Text Annotation

Text Annotations are a mechanism for a modeler to provide additional information for the reader of a BPMN Diagram.

◆ A Text Annotation is an open rectangle that MUST be drawn with a solid single black line (as seen in Figure 9.42).

◆ The use of text, color, size, and lines for a Text Annotation MUST follow the rules defined in Section 8.3, "Use of Text, Color, Size, and Lines in a Diagram," on page 29.

The Text Annotation object can be connected to a specific object on the Diagram with an Association (see Figure 9.42), but do not affect the flow of the Process. Text associated with the Annotation can be placed within the bounds of the open rectangle.

Text Annotation Allows
a Modeler to provide
additional Information
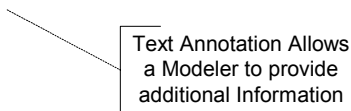
Figure 9.42 - A Text Annotation

Attributes

The following table displays the attributes for Annotations, which extends the set of common Artifact attributes (see Table 9.41). These attributes only apply if the ArtifactType attribute is set to Annotation:

Table 9.43 - Text Annotation Attributes

Attributes	Description
Text : String	Text is an attribute that is text that the modeler wishes to communicate to the reader of the Diagram.

9.7.4 Group

The Group object is an Artifact that provides a visual mechanism to group elements of a diagram informally. The grouping is tied to the Category supporting element (which is an attribute of all BPMN elements). That is, a Group is a visual depiction of a single Category. The graphical elements within the Group will be assigned the Category of the Group. (Note -- Categories can be highlighted through other mechanisms, such as color, as defined by a modeler or a modeling tool).

◆ A Group is a rounded corner rectangle that MUST be drawn with a solid dashed black line (as seen in Figure 9.43).

◆ The use of text, color, size, and lines for a Group MUST follow the rules defined in Section 8.3, "Use of Text, Color, Size, and Lines in a Diagram," on page 29.

Figure 9.43 - A Group Artifact

As an Artifact, a Group is not an activity or any Flow Object, and, therefore, cannot connect to Sequence Flow or Message Flow. In addition, Groups are not constrained by restrictions of Pools and Lanes. This means that a Group can stretch across the boundaries of a Pool to surround Diagram elements (see Figure 9.44), often to identify activities that exist within a distributed business-to-business transaction.
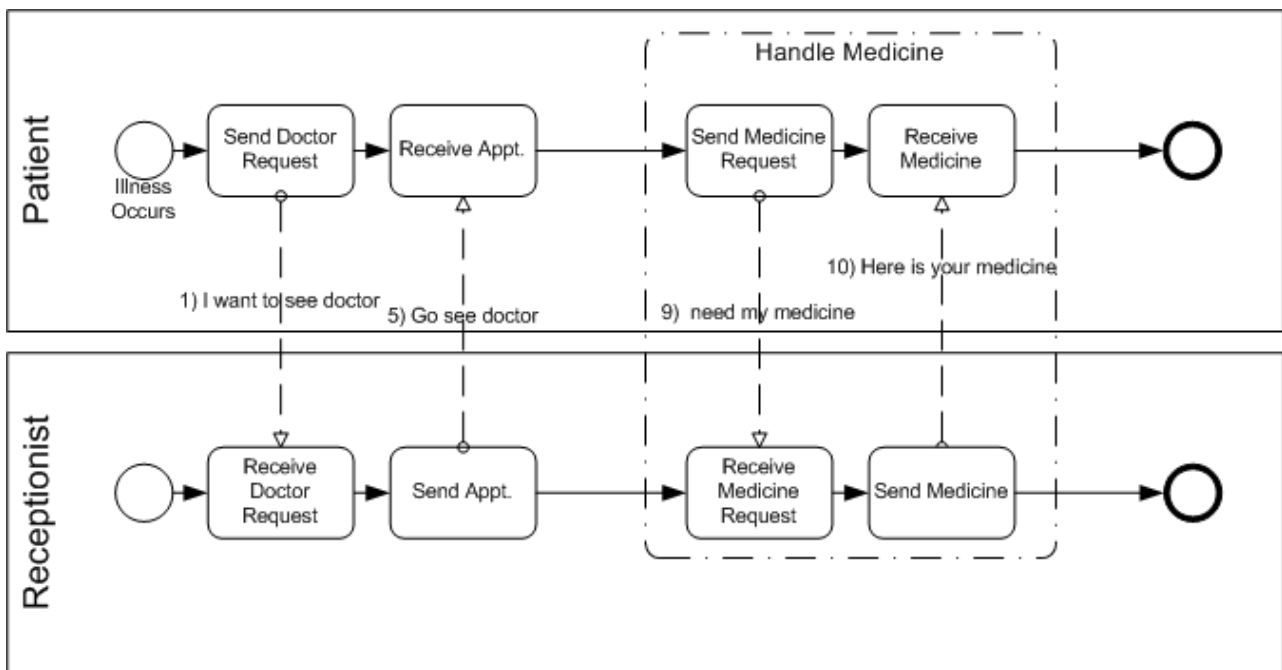
Figure 9.44 - A Group around activities in different Pools

Groups are often used to highlight certain sections of a Diagram without adding additional constraints for performance--as a Sub-Process would. The highlighted (grouped) section of the Diagram can be separated for reporting and analysis purposes. Groups do not affect the flow of the Process.

Attributes

The following table displays the attributes for Groups, which extends the set of common Artifact attributes (see Table 9.41). These attributes only apply if the ArtifactType attribute is set to Group:

Table 9.44 - Group Attributes

Attributes	Description
CategoryRef : Category	CategoryRef specifies the Category that the Group represents (Further details about the definition of a Category can be found in "See "Category" on page 277.."). The name of the Category provides the label for the Group. The graphical elements within the boundaries of the Group will be assigned the Category.
GraphicalElements (0-n) : Graphical Element	The GraphicalElements attribute identifies all of the graphical elements (e.g., Events, Activities, Gateways, and Artifacts) that are within the boundaries of the Group.

10   Business Process Diagram Connecting Objects

This section defines the graphical objects used to connect two objects together (i.e., the connecting lines of the Diagram) and how the flow progresses through a Process (i.e., through a straight sequence or through the creation of parallel or alternative paths).

10.1   Graphical Connecting Objects

There are two ways of Connecting Objects in BPMN: a Flow, either sequence or message, and an Association. Sequence Flow and Message Flow, to a certain extent, represent orthogonal aspects of the business processes depicted in a model, although they both affect the performance of activities within a Process. In keeping with this, Sequence Flow will generally flow in a single direction (either left to right, or top to bottom) and Message Flow will flow at a 90° from the Sequence Flow. This will help clarify the relationships for a Diagram that contains both Sequence Flow and Message Flow. However, BPMN does not restrict this relationship between the two types of Flow. A modeler can connect either type of Flow in any direction at any place in the Diagram.

The next three sections will describe how these types of connections function in BPMN.

10.1.1 Common Connecting Object Attributes

The following table displays the set of attributes common to Connecting Objects (Sequence Flow, Message Flow, and Association), and which extends the set of common BPMN Element attributes (see Table 9.1):

Table 10.1 - Common Connecting Object Attributes

Attributes	Description
Name (0-1) : String	Name is an optional attribute that is text description of the Connecting Object.
SourceRef : Graphical Element	SourceRef is an attribute that identifies which Graphical Element the Connecting Object is connected *from*. Note: there are restrictions as to what objects Sequence Flow and Message Flow can connect. Refer to the Sequence Flow Connections section and the Message Flow Connections section for each Flow Object, Swimlane, and Artifact.
TargetRef : Graphical Element	TargetRef is an attribute that identifies which Graphical Element the Connecting Object is connected *to*. Note: there are restrictions as to what objects Sequence Flow and Message Flow can connect. Refer to the Sequence Flow Connections section and the Message Flow Connections section for each Flow Object, Swimlane, and Artifact.

10.1.2 Sequence Flow

A Sequence Flow is used to show the order that activities will be performed in a Process. Each Flow has only one source and only one target. The source and target must be from the set of the following Flow Objects: Events (Start, Intermediate, and End), Activities (Task and Sub-Process), and Gateways. During performance (or simulation) of the process, a Token will leave the source Flow Object, traverse down the Sequence Flow, and enter the target Flow Object.

◆ A Sequence Flow is line with a solid arrowhead that MUST be drawn with a solid single line (as seen in Figure 10.1).

    ◆ The use of text, color, and size for Sequence Flow MUST follow the rules defined in Section 8.3, "Use of Text, Color, Size, and Lines in a Diagram," on page 29.
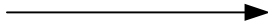
Figure 10.1 - A Sequence Flow

BPMN does not use the term "Control Flow" when referring the lines represented by Sequence Flow or Message Flow. The start of an activity is "controlled" not only by Sequence Flow (the order of activities), but also by Message Flow (a message arriving), as well as other process factors, such as scheduled resources. Artifacts can be Associated with activities to show some of these other factors. Thus, we are using a more specific term, "Sequence Flow," since these lines mainly illustrate the sequence that activities will be performed.

◆ A Sequence Flow MAY have a conditional expression attribute, depending on its source object.

This means that the condition expression must be evaluated before a Token can be generated and then leave the source object to traverse the Flow. The conditions are usually associated with Decision Gateways, but can also be used with activities.

◆ If the source of the Sequence Flow is an activity, rather than Gateway, then a Conditional Marker, shaped as a "mini-diamond"," MUST be used at the beginning of the Sequence Flow (see Figure 10.2).

The diamond shape is used to relate the behavior to a Gateway (also a diamond) that controls the flow within a Process. More information about how conditional Sequence Flow are used can be found in "Splitting Flow" on page 115.
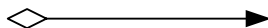
Figure 10.2 - A Conditional Sequence Flow

A Sequence Flow that has an Exclusive Data-Based Gateway or an activity as its source can also be defined with a condition expression of Default. Such Sequence Flow will have a marker to show that it is a Default flow.

◆ The Default Marker MUST be a backslash near the beginning of the line (see Figure 10.3).
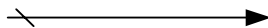
Figure 10.3 - A Default Sequence Flow

Attributes

The following table displays the set of attributes of a Sequence Flow, and which extends the set of common Connecting Object attributes (see Figure 10.43):

Table 10.2 - Sequence Flow Attributes

Attributes	Description
ConditionType (None | Expression | Default) None : String	By default, the ConditionType of a Sequence Flow is None. This means that there is no evaluation at runtime to determine whether or not the Sequence Flow will be used. Once a Token is ready to traverse the Sequence Flow (i.e., the Source is an activity that has completed), then the Token will do so. The normal, uncontrolled use of Sequence Flow, in a sequence of activities, will have a None ConditionType (see Figure 10.11). A None ConditionType MUST NOT be used if the Source of the Sequence Flow is an Exclusive Data-Based or Inclusive Gateway. The ConditionType attribute MAY be set to Expression if the Source of the Sequence Flow is a Task, a Sub-Process, or a Gateway of type Exclusive-Data-Based or Inclusive. If the ConditionType attribute is set to Expression, then a condition marker SHALL be added to the line if the Sequence Flow is outgoing from an activity (see Figure 10.2). However, a condition indicator MUST NOT be added to the line if the Sequence Flow is outgoing from a Gateway. An Expression ConditionType MUST NOT be used if the Source of the Sequence Flow is an Event-Based Exclusive Gateway, a Complex Gateway, a Parallel Gateway, a Start Event, or an Intermediate Event. In addition, an Expression ConditionType MUST NOT be used if the Sequence Flow is associated with the Default Gate of a Gateway. The ConditionType attribute MAY be set to Default only if the Source of the Sequence Flow is an activity or an Exclusive Data-Based Gateway. If the ConditionType is Default, then the Default marker SHALL be displayed (see Figure 10.3).
[ConditionType is set to Expression only] **ConditionExpression**: Expression	If the ConditionType attribute is set to Expression, then the ConditionExpression attribute MUST be defined as a valid expression. The expression will be evaluated at runtime. If the result of the evaluation is TRUE, then a Token will be generated and will traverse the Sequence--Subject to any constraints imposed by a Source that is a Gateway.

10.1.3 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. Thus,

◆ Message Flow MUST connect two Pools, either to the Pools themselves or to Flow Objects within the Pools. They cannot connect two objects within the same Pool.

◆ A Message Flow is line with a open arrowhead that MUST be drawn with a dashed single black line (as seen in

Figure 10.4).

◆ The use of text, color, size, and lines for Message Flow MUST follow the rules defined in Section 8.3, "Use of Text, Color, Size, and Lines in a Diagram," on page 29.

Figure 10.4 - A Message Flow

The Message Flow can connect directly to the boundary of a Pool (See Figure 10.5), especially if the Pool does not have any process details within (e.g., is a "Black Box").
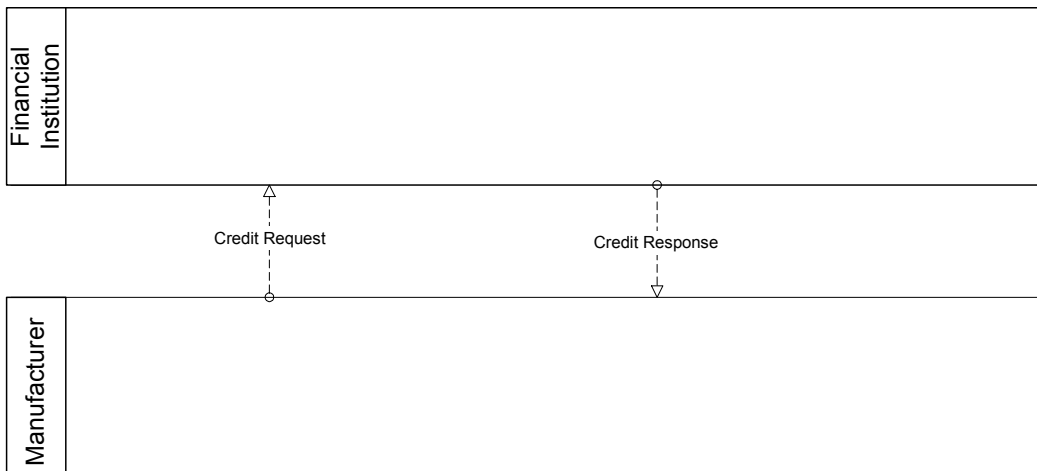
Figure 10.5 - Message Flow connecting to the boundaries of two Pools

A Message Flow can also cross the boundary of a Pool and connect to a Flow Object within that Pool (see Figure 10.6).
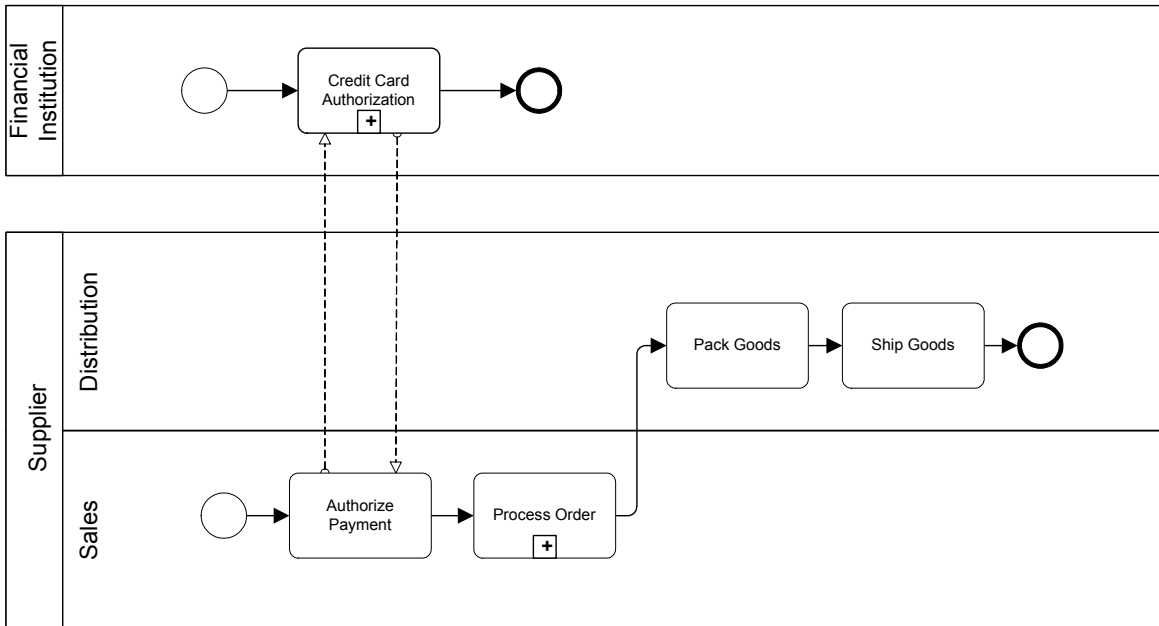
Figure 10.6 - Message Flow connecting to Flow Objects within two Pools

If there is an Expanded Sub-Process in one of the Pools, then the message flow can be connected to either the boundary of the Sub-Process or to objects within the Sub-Process.

Attributes

The following table displays the identified attributes of a Message Flow, and which extends the set of common Connecting Object attributes (see Table 10.1):

Table 10.3 - Message Flow Attributes

Attributes	Description
MessageRef (0-1) : Message	MessageRef is an optional attribute that identifies the Message that is being sent. The attributes of a Message can be found in Section B.11.11, "Message," on page 283.

10.1.4 Association

An Association is used to associate information and Artifacts with Flow Objects. Text and graphical non-Flow Objects can be associated with the Flow Objects and Flow. An Association is also used to show the activities used to compensate for an activity. More information about compensation can be found in Section 10.3, "Compensation Association," on page 133.

  ◆ An Association Flow is line that MUST be drawn with a dotted single black line (as seen in Figure 10.7).

   ◆ The use of text, color, size, and lines for an Association MUST follow the rules defined in Section 8.3, "Use of Text, Color, Size, and Lines in a Diagram," on page 29.

Figure 10.7 - An Association

If there is a reason to put directionality on the association then:

◆ A line arrowhead MAY be added to the Association line. (see Figure 10.8).

A directional Association is often used with Data Objects to show that a Data Object is either an input to or an output from an activity.

---------------------≫
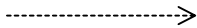
Figure 10.8 - A directional Association

An Association is used to connect user-defined text (an Annotation) with a Flow Object (see Figure 10.9).

```
┌──────────────┐
│  Announce    │
│  Issues for  │
│  Discussion  │
└──────────────┘
        ┆
        ┆
        ┆  ┌─────────────────────────
        ┆  │ Allow 1 week for the
        ┆  │ discussion of the Issues
           │ — through e-mail or
           │         calls
           └─────────────────────────
```
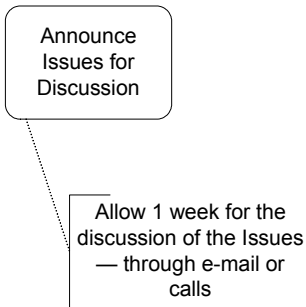
Figure 10.9 - An Association of Text Annotation

An Association is also used to associate Data Objects with other objects (see Figure 10.10). A Data Object is used to show how documents are used throughout a Process. See Section 9.7.2, "Data Object," on page 95 for more information on Data Objects.
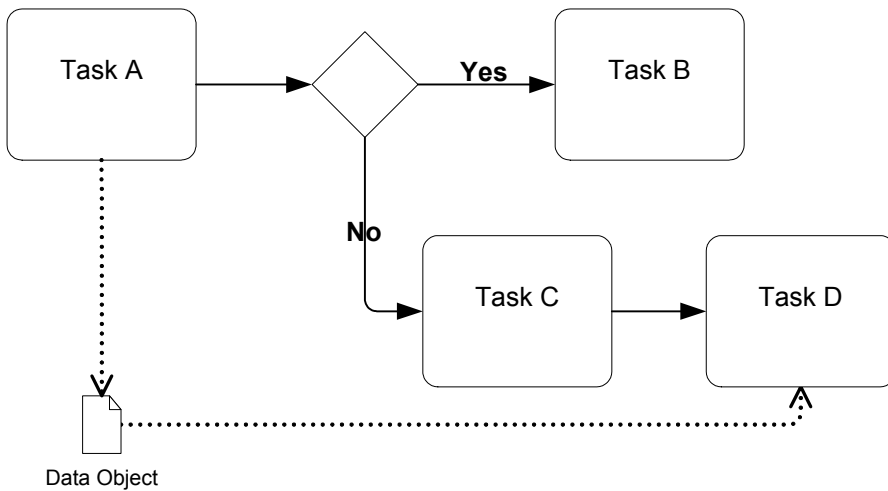
Figure 10.10 - An Association connecting a Data Object with a Flow

Attributes

The following table displays the identified attributes of a Association, and which extends the set of common Connecting Object attributes (see Table 10.1):

Table 10.4 - Association Attributes

Attributes	Description
Direction (None | One | Both) None : String	Direction is an attribute that defines whether or not the Association shows any directionality with an arrowhead. The default is None (no arrowhead). A value of One means that the arrowhead SHALL be at the Target Object. A value of Both means that there SHALL be an arrowhead at both ends of the Association line.

10.2 Sequence Flow Mechanisms

The Sequence Flow mechanisms described in the following sections are divided into four types: Normal, Exception, Link Events, and Ad Hoc (no flow). Within these types of flow, BPMN can be related to specific "Workflow Patterns[1]." These patterns began as development work by Wil van der Aalst, Arthur ter Hofstede, Bartek Kiepuszewski, and Alistair Barros[2]. Twenty-one patterns have been defined as a way to document specific behavior that can be executed by a BPM system. These patterns range from very simple behavior to very complex business behavior. These patterns are useful in that they provide a comprehensive checklist of behavior that should be accounted for by BPM system. Therefore, some of these patterns will be illustrated with BPMN in the following sections to show how BPMN can handle the simple and complex requirements for Business Process Modeling.

---

1. http://tmitwww.tm.tue.nl/research/patterns/
2. http://tmitwww.tm.tue.nl/research/patterns/download/wfs-pat-2002.pdf

10.2.1 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. The simplest type of flow within a Process is a sequence, which defines a dependencies of order for a series of activities that will be performed (sequentially). A sequence is also Workflow Pattern #1 -- Sequence[3] (see Figure 10.11).
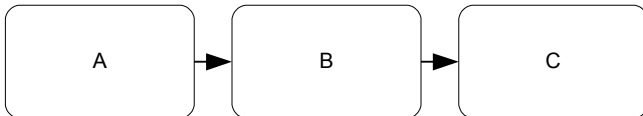
Figure 10.11 - Workflow Pattern #1: Sequence

As stated previously, the normal Sequence Flow should be completely exposed and no flow behavior hidden. This means that a viewer of a BPMN Diagram will be able to trace through a series of Flow Objects and Sequence Flow, from the beginning to the end of a given level of the Process without any gaps or hidden "jumps" (see Figure 10.12). In this figure, Sequence Flow connect all the objects in the Diagram, from the Start Event to the End Event. The behavior of the Process shown will reflect the connections as shown and not skip any activities or "jump" to the end of the Process.
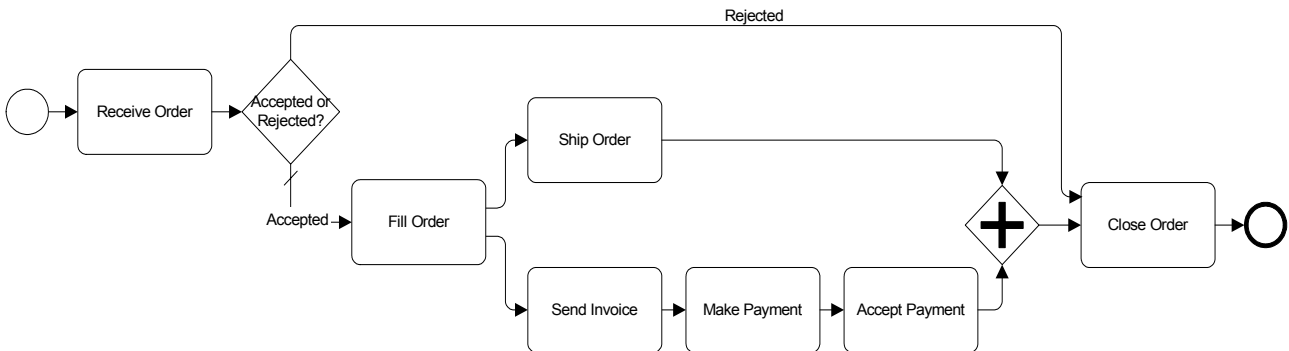
Figure 10.12 - A Process with Normal Flow

As the Process continues through the series of Sequence Flow, control mechanisms may divide or combine the Sequence Flow as a means of describing complex behavior. There are control mechanisms for dividing (forking and splitting) and for combining (joining and merging) Sequence Flow. Gateways and conditional Sequence Flow are used to accomplish the dividing and combining of flow. It is possible that there may be gaps in the Sequence Flow if Gateways and/or conditional Sequence Flow are not configured to cover all performance possibilities. In this case, a model that violates the flow traceability requirement will be considered an invalid model. Presumably, process development software or BPM test environments will be able to test a process model to ensure that the model is valid.

---

3. http://tmitwww.tm.tue.nl/research/patterns/sequence.htm

A casual look at the definitions of the English terms for these mechanisms (e.g., forking and splitting) would indicate that each pair of terms mean basically the same thing. However, their effect on the behavior of a Process is quite different. We will continue to use these English terms but will provide specific definitions about how they affect the performance of the process in the next few sections of this specification.

The use of an expanded Sub-Process in a Process (see Figure 10.13), which is the inclusion of one level of the Process within another Level of the Process, can sometimes break the traceability of the flow through the lines of the Diagram. The Sub-Process is not required to have a Start Event and an End Event. This means that the series of Sequence Flow will be disrupted from border of the Expanded Sub-Process to the first object within the Expanded Sub-Process. The flow will "jump" to the first object within the Expanded Sub-Process. Expanded Sub-Processes will often be used, as seen in the figure, to include exception handling. A requirement that modelers always include a Start Event and End Event within Expanded Sub-Processes would mainly add clutter to the Diagram without necessarily adding to the clarity of the Diagram. Thus, BPMN does not require the use of Start Events and End Events to satisfy the traceability of a Diagram that contains multiple levels.
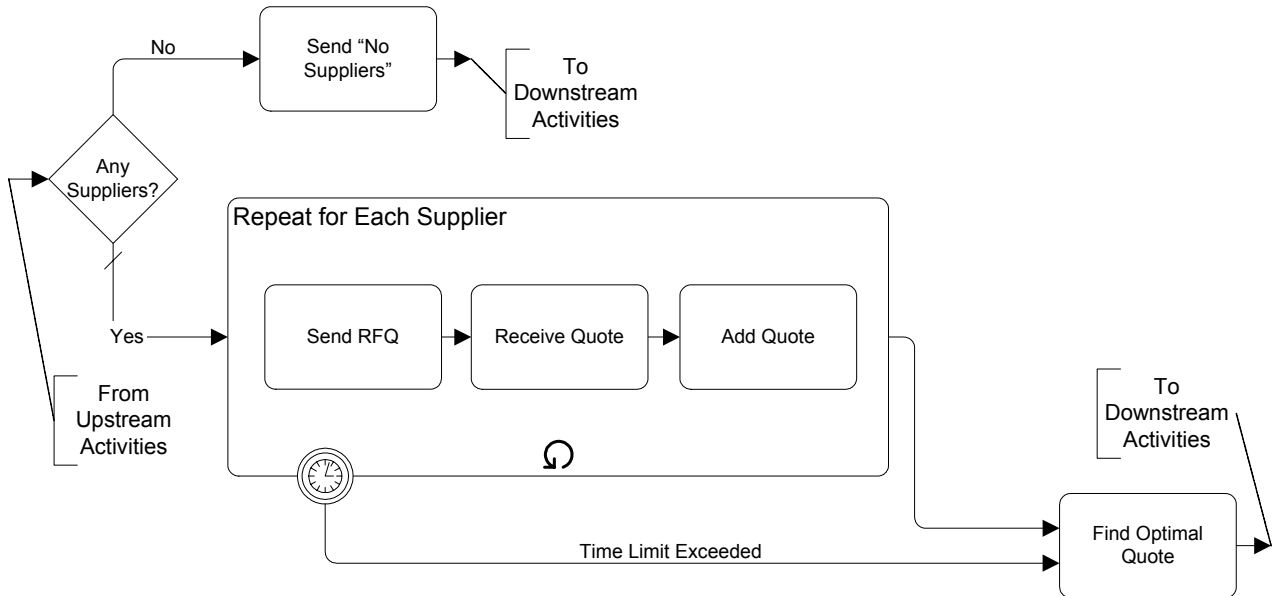
Figure 10.13 - An Expanded Sub-Process without a Start Event and End Event

Of course, the Start and End Events for an Expanded Sub-Process can be included and placed entirely within its boundaries (see Figure 10.14). This type of model will also have a break from a completely traceable Sequence Flow as the flow continues from one Process level to another.
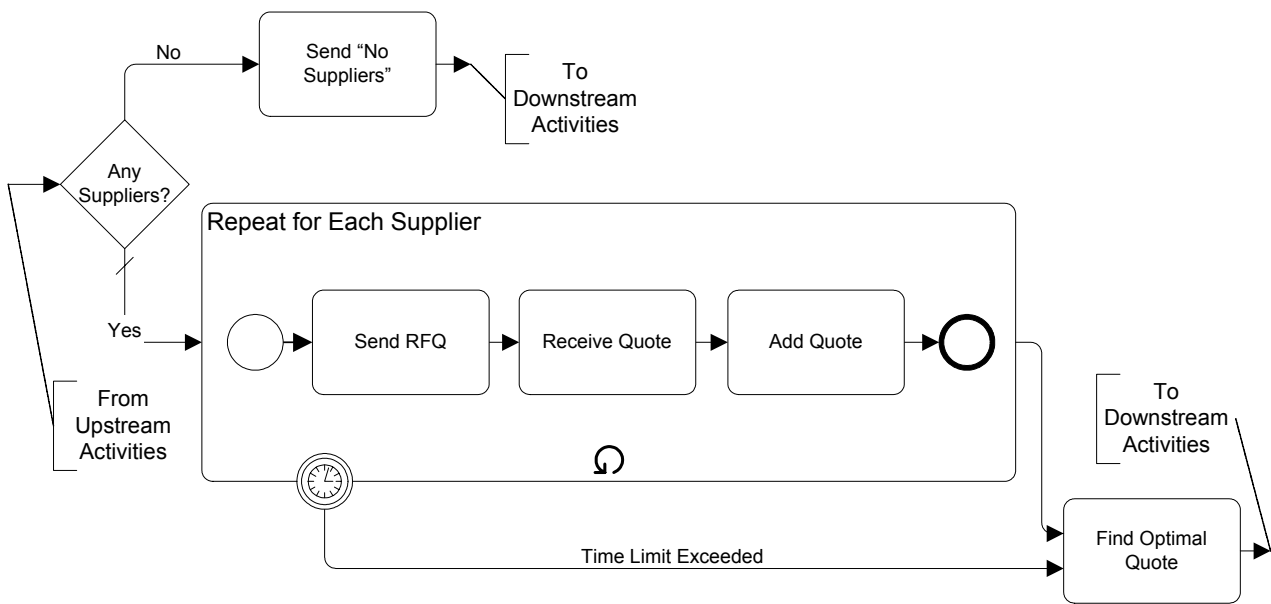
Figure 10.14 - An Expanded Sub-Process with a Start Event and End Event Internal

However, a modeler may want to ensure the traceability of a Diagram and can use a Start Event and End Event in an Expanded Sub-Process. One way to do this would be to attach these events to the boundary of the Expanded Sub-Process (see Figure 10.15). The incoming Sequence Flow to the Sub-Process can be attached directly to the Start Event instead of the boundary of the Sub-Process. Likewise, the outgoing Sequence Flow from the Sub-Process can connect from the End Event instead of the boundary of the Sub-Process. Doing this, the Normal Flow can be traced throughout a multi-level Process.

Technically, the Start and End Events still reside within the Sub-Process. The use of this modeling technique is just a graphical short-cut to a more accurate depiction of the Process (i.e., as shown in Figure 10.14. Therefore, the Sequence Flow connecting to the Start Event and connecting from the End Event do not violate the Sequence Flow connection rules (as defined in "Sequence Flow Connections" on page 39 and "Sequence Flow Connections" on page 43).
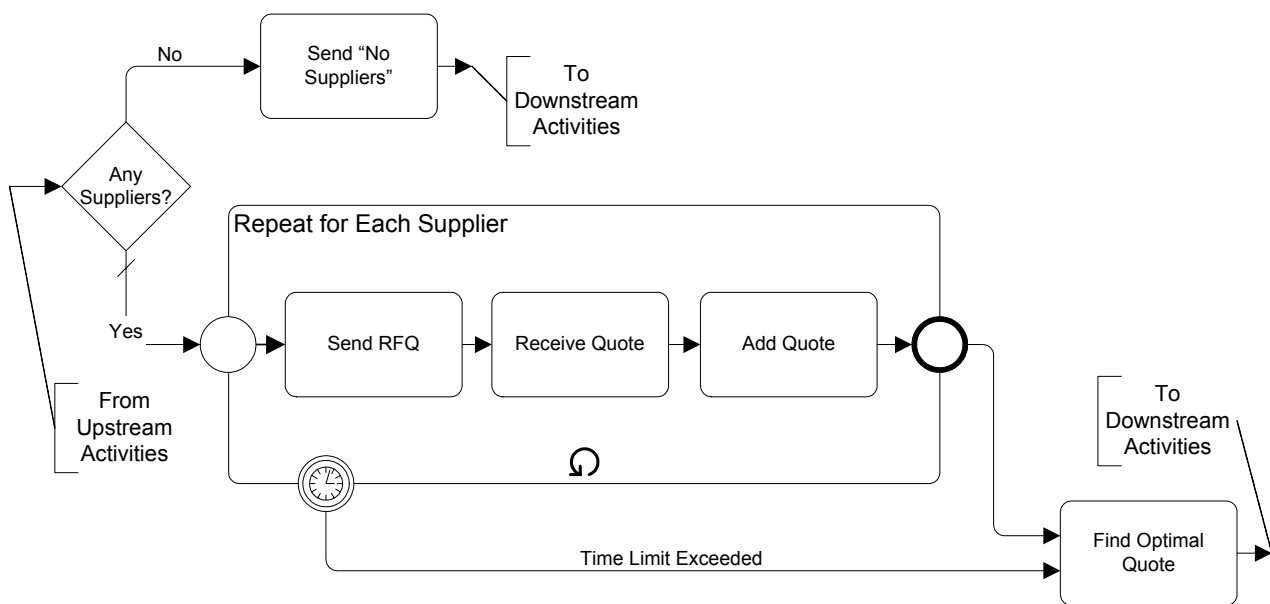
Figure 10.15 - An Expanded Sub-Process with a Start Event and End Event Attached to Boundary

When dealing with Exceptions and Compensation, the traceability requirement is also relaxed (Section 10.2.2, "Exception Flow," on page 131 and Section 10.3, "Compensation Association," on page 133).

Forking Flow

BPMN uses the term forking to refer to the dividing of a path into two or more parallel paths (also known as an AND-Split). It is a mechanism that will allow activities to be performed concurrently, rather than sequentially. This is also Workflow Pattern #2 -- Parallel Split[4]. BPMN provides three configurations that provide forking.

The first mechanism to create a fork is simple: a Flow Object can have two or more outgoing Sequence Flow (see Figure 10.16). A special flow control object is not used to fork the path in this case, since it is considered uncontrolled flow; that is, flow will proceed down each path without any dependencies or conditions--there is no Gateway that controls the flow. Forking Sequence Flow can be generated from a Task, Sub-Process, or a Start Event.

---

4.   http://tmitwww.tm.tue.nl/research/patterns/parallel_split.htm
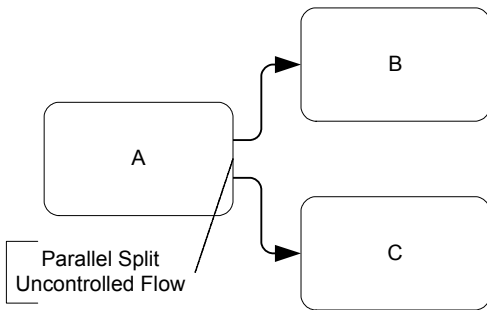
Figure 10.16 - Workflow Pattern #2: Parallel Split -- Version 1

The second mechanism uses a Parallel Gateway (see Figure 10.20). For situations as shown in the Figure 10.17, a Gateway is not needed, since the same behavior can be created through multiple outgoing Sequence Flow, as in Figure 10.16. However, some modelers and modeling tools may use a forking Gateway as a "best practice." See Section 9.5.5, "Parallel Gateways," on page 86 for more information on Parallel Gateways.
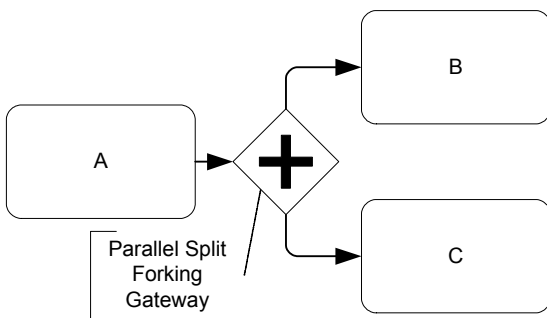
Figure 10.17 - Workflow Pattern #2: Parallel Split -- Version 2

Even when not required as a "best practice," there are situations were the Parallel Gateway provides a useful indicator of the behavior of the Process. Figure 10.18 shows how a forking Gateway is used when the output of an *Exclusive* Decision requires that multiple activities will be performed based on one condition (Gate).
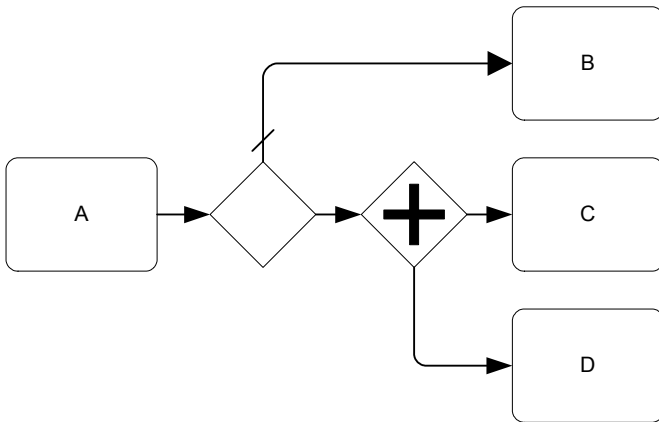
Figure 10.18 - The Creation of Parallel Paths with a Gateway

While multiple conditional Sequence Flow, each with the exact same condition expression (see Figure 10.19), could be used with an *Inclusive* Gateway to create the behavior, the use of a forking Gateway makes the behavior much more obvious.
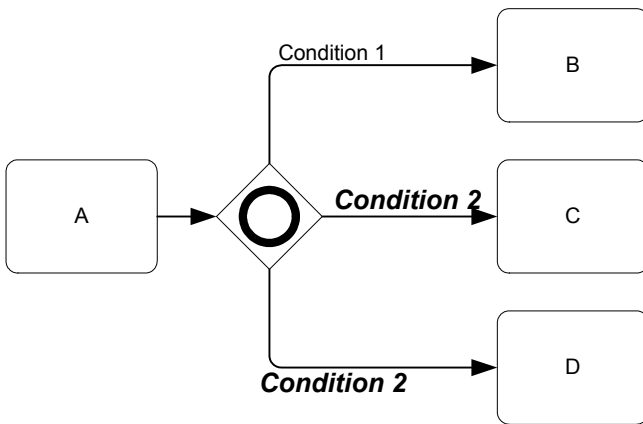
Figure 10.19 - The Creation of Parallel Paths with Equivalent Conditions

This third version of the forking mechanism uses an Expanded Sub-Process to group a set of activities to be performed in parallel (see Figure 10.20). The Sub-Process does not include a Start and End Event and displays the activities "floating" within. A configuration like this can be called a "parallel box" and can be a compact and less cluttered way of showing parallelism in the Process. The capability to model in this way is the reason that Start and End Events are optional in BPMN.
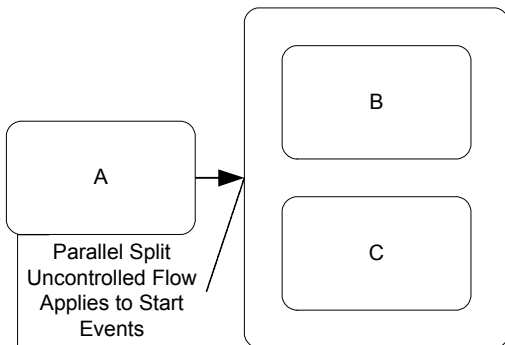
Figure 10.20 - Workflow Pattern #2: Parallel Split -- Version 3

Most of the time, the paths that have been divided with a fork are combined back together through a join (refer to the next section) and synchronized before the flow will continue. However, BPMN provides the flexibility for advanced methods to handle complex process situations. Thus, the exact behavior will be determined by the configuration of the Sequence Flow and the Gateways that are used.

Joining Flow

BPMN uses the term joining to refer to the combining of two or more parallel paths into one path (also known as an AND-Join). A Parallel Gateway is used to synchronize two or more incoming Sequence Flow (see Figure 10.21). In general, this means that Tokens created at a fork will travel down parallel paths and then meet at the Parallel Gateway. From there, only one Token will continue. This is also Workflow Pattern #3 -- Synchronization[5]. Section 9.5.5, "Parallel Gateways," on page 86 for more information on Parallel Gateways.
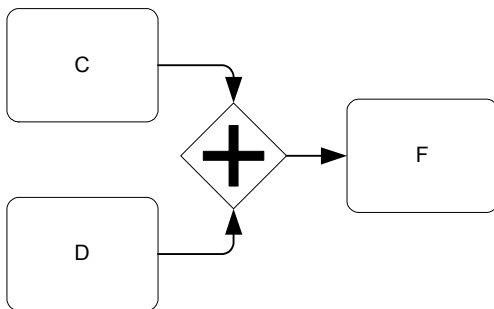
Figure 10.21 - Workflow Pattern #3: Synchronization -- Version 1

Another mechanism for synchronization is the completion of a Sub-Process (see Figure 10.22). If there are parallel paths within the Sub-Process that are *not* synchronized with an Parallel Gateway, then they will eventually reach an End Event (even if the End Event is implied). The default behavior of a Sub-Process is to wait until all activity within has been completed before the flow will move back up to a higher level Process. Thus, the completion of a Sub-Process is a synchronization point.

---

5.   http://tmitwww.tm.tue.nl/research/synchronization.htm
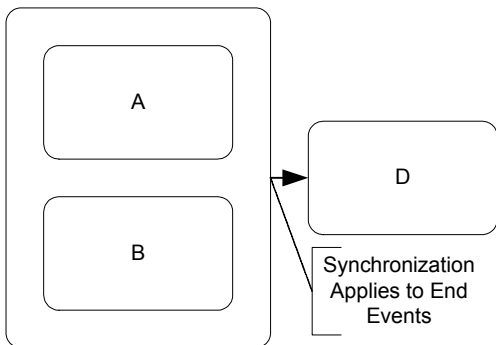
Figure 10.22 - Workflow Pattern #3: Synchronization -- Version 2

There is no specific correlation between the joining of a set of parallel paths and the forking that created the parallel paths. For example, an activity may have three outgoing Sequence Flow, which creates a fork of three parallel paths, but these three paths do not need to be joined at the same object. Figure 10.23 shows that two of three parallel paths are joined at Task "F." All of the paths eventually will be joined, but this can happen through any combination of objects, including lone End Events. In fact, each path could end with a separate End Event, and then be synchronized as mentioned above.
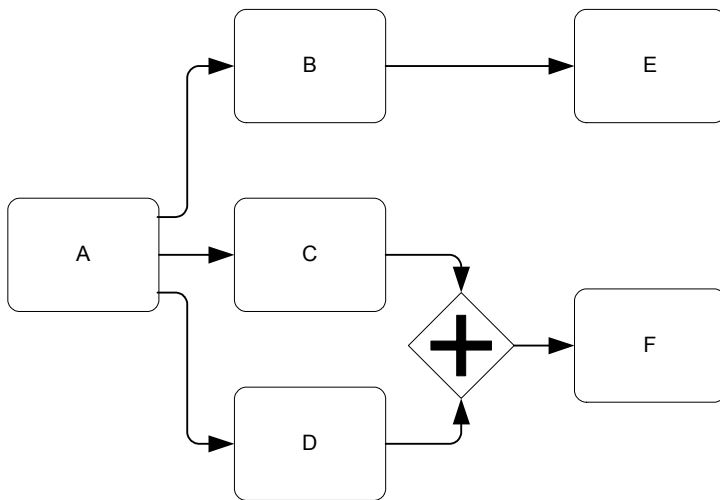
Figure 10.23 - The Fork-Join Relationship is not Fixed

Splitting Flow

BPMN uses the term splitting to refer to the dividing of a path into two or more alternative paths (also known as an OR-Split). It is a place in the Process where a question is asked, and the answer determines which of a set of paths is taken. It is the "fork in the road" where a traveler, in this case a Token, can take only one of the forks (not to be confused with forking—see below).

The general concept of splitting the flow is usually referring to as a Decision. In traditional flow charting methodologies, Decisions are depicted as diamonds and usually are exclusive. BPMN also uses a diamond to leverage the familiarity of the shape, but extends the use of the diamond to handle the complex behavior of business processes (which cannot be handled by traditional flow charts). The diamond shape is used in both Gateways and the beginning of a conditional Sequence Flow (when exiting an activity). Thus, when readers of BPD see a diamond, they know that the flow will be controlled in some way and will not just pass from one activity to another. The location of the mini-diamond and the internal indicators within the Gateways will indicate how the flow will be controlled.

There are multiple configurations to split the flow within BPMN so that different types of complex behavior can be modeled. Conditional Sequence Flow and three types of Gateways (Exclusive, Inclusive, and Complex) are used to split the flow. See Section 10.1.2, "Sequence Flow," on page 101 for details on conditional Sequence Flow. Section 9.5, "Gateways," on page 71 for details on the Gateways.

There are two basic mechanism for making the Decision during the performance of the Process: the first is an evaluation of a condition expression. There are three variations of this mechanism: Exclusive, Inclusive, and Complex. The first variation, an Exclusive Decision, is the same as Workflow Pattern #4 -- Exclusive Choice[6] (see Figure 10.24). See "Data-Based" on page 74 for more information on Data-Based Exclusive Gateways.
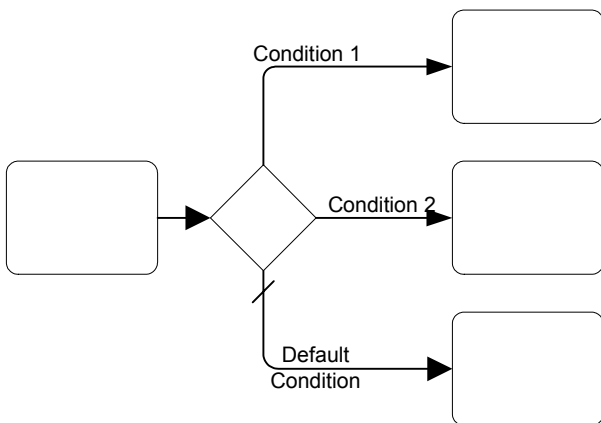
Figure 10.24 - A Data-Based Decision Example -- Workflow Pattern #4 -- Exclusive Choice

The second type of expression evaluation is the Inclusive Decision, which is also Workflow Pattern #6 -- Multiple Choice[7]. There are two configurations of the Inclusive Decision. The first type of Inclusive Decisions uses conditional Sequence Flow from an Activity (see Figure 10.25).

---

6.   http://tmitwww.tm.tue.nl/research/patterns/exclusive_choice.htm
7.   http://tmitwww.tm.tue.nl/research/patterns/multiple_choice.htm
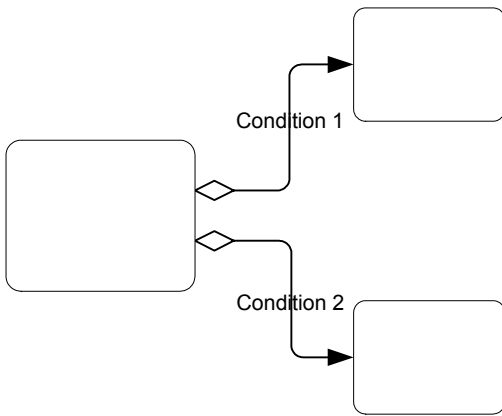
Figure 10.25 - Workflow Pattern #6 -- Multiple Choice -- Version 1

The second type of Inclusive Decisions uses an Inclusive Gateway to control the flow (see Figure 10.26). See Section 9.5.3, "Inclusive Gateways," on page 81 for more information on Inclusive Gateways.
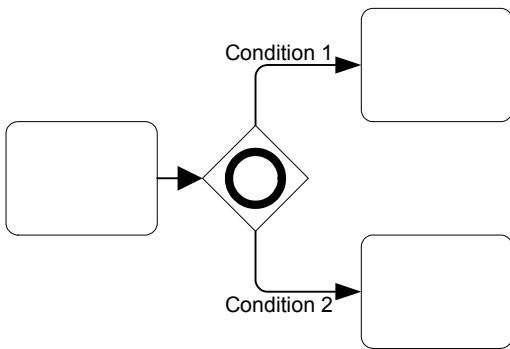
Figure 10.26 - Workflow Pattern #6 -- Multiple Choice -- Version 2

The third type of expression evaluation is the Complex Decision (see Figure 10.27). See Section 9.5.4, "Complex Gateways," on page 84 for more information on Complex Gateways.
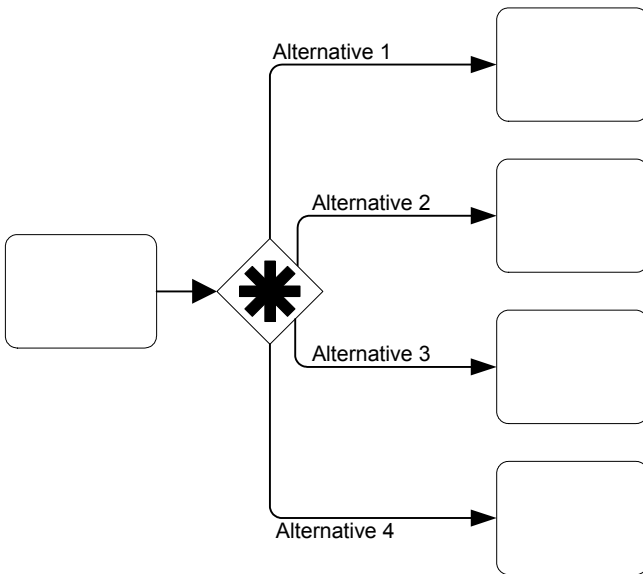
Figure 10.27 - A Complex Decision (Gateway)

The second mechanism for making a Decision is the occurrence of a particular event, such as the receipt of a message (see Figure 10.28). See "Event-Based" on page 78 for more information on Event-Based Exclusive Gateways.
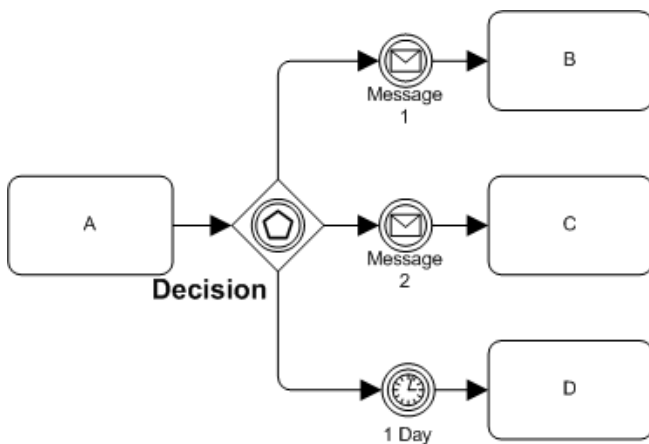
Figure 10.28 - An Event-Based Decision Example

Merging Flow

BPMN uses the term merging to refer to the combining of two or more alternative paths into one path (also known as an a OR-Join). It is a place in the process where two or more alternative paths begin to traverse activities that are common to each of the paths. Theoretically, each alternative path can be modeled separately to a completion (an End Event). However, merging allows the paths to overlap and avoids the duplication of activities that are common to the separate paths. For a given instance of the Process, a Token would actually only see the sequence of activities that exist in one of the paths as if it were modeled separately to completion.

Since there are multiple ways that Sequence Flow can be forked and split, there are multiple ways that Sequence Flow can be merged. There are five different Workflow Patterns that can be demonstrated with merging.

The first Workflow Pattern, Simple Merge[8], The graphical mechanism to merge alternative paths is simple: there are two or more incoming Sequence Flow to a Flow Object (see Figure 10.29). In general, this means that a Token will travel down one of the alternative paths (for a given Process instance) and will continue from there. For that instance, Tokens will never arrive down the other alternative paths. BPMN provides two versions of a Simple Merge.

The first version is shown in Figure 10.29. The two incoming Sequence Flow for activity "D" are uncontrolled. Since the two Sequence Flow are at the end of two alternative paths, created through the upstream exclusive Gateway, only one Token will reach activity "D" for any given instance of the Process.
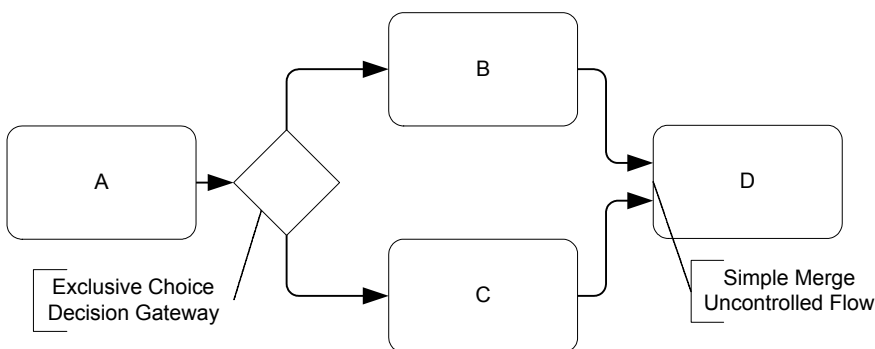
Figure 10.29 - Workflow Pattern #5 -- Simple Merge – Version 1

If the multiple incoming Sequence Flow are actually parallel instead of alternative, then the end result is different, even though the merging configuration is the same as Figure 10.29. In Figure 10.30, the upstream behavior is parallel. Thus, there will be two Tokens arriving (at different times) at activity "D." Since the flow into activity "D" in uncontrolled, *each Token arriving at activity "D" will cause a new instance of that activity*. This is an important concept for modelers of BPMN should understand. In addition, this type of merge is the Workflow Pattern Multiple Merge[9].
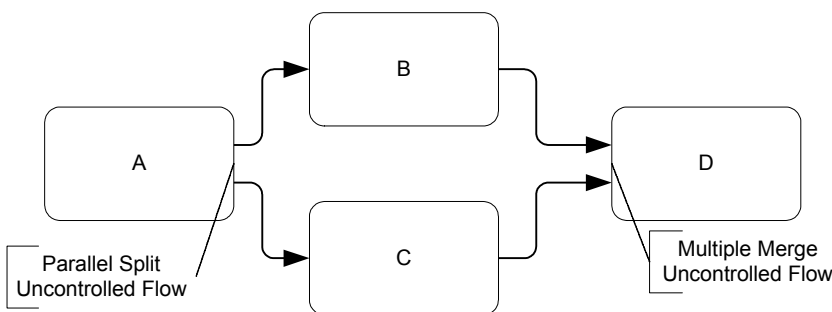
Figure 10.30 - Workflow Pattern #7 -- Multiple Merge

---

8.   http://tmitwww.tm.tue.nl/research/patterns/simple_merge.htm
9.   http://tmitwww.tm.tue.nl/research/patterns/multiple_merge.htm

The second version of the Simple Merge is shown in Figure 10.31. The two incoming Sequence Flow for activity "D" are controlled through the Exclusive Gateway. Since the two Sequence Flow are at the end of two alternative paths, created through the upstream exclusive Gateway, only one Token will reach the Gateway for any given instance of the Process. The Token will then immediately proceed to activity "D."
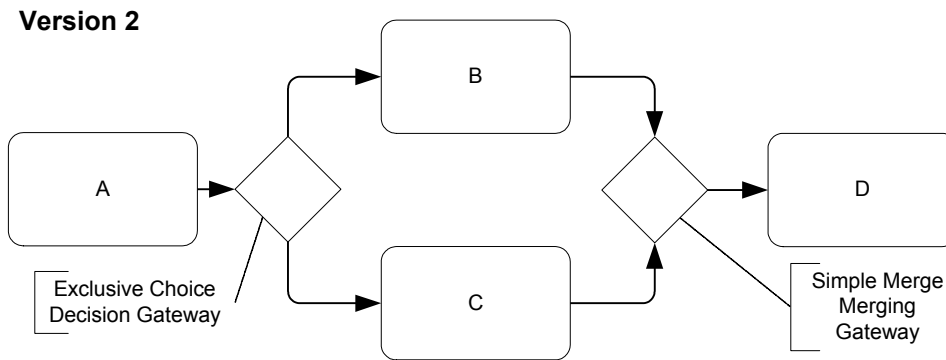
Version 2

Figure 10.31 - Workflow Pattern #5 -- Simple Merge – Version 2

Another merging situation is the Workflow Pattern Discriminator[10]. In this situation, the multiple incoming Sequence Flow are parallel instead of alternative (see Figure 10.32). Thus, there will be two Tokens arriving (at different times) at the Complex Gateway preceding activity "D." To satisfy the Discriminator pattern, the Complex Gateway must accept the first Token and immediately pass it on through to the activity. When the second Token arrives, it will be *excluded* from the remainder of the flow. This means that the Token will not be passed on to the activity, but will be consumed.
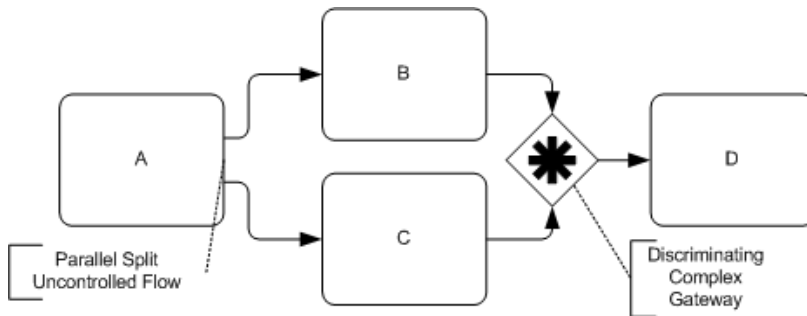
Figure 10.32 - Workflow Pattern #8 -- Discriminator

The fourth type of Workflow Pattern merge is called a Synchronizing Join[11]. This is a situation when the merging location does not know ahead of time how many Tokens will be arriving at the Gateway. In some Process instances, there may be only one Token. In other Process instances, there may be more than one Token arriving. This type of situation is created when an Inclusive Decision is made up stream (see Figure 10.33). To handle this, an Inclusive Gateway can be used to

---

10. http://tmitwww.tm.tue.nl/research/patterns/discriminator.htm
11. http://tmitwww.tm.tue.nl/research/patterns/synchronizing_join.htm

merge the appropriate number of Tokens for each Process instance. The Gateway, following the pattern Synchronizing Join, will wait for all expected Tokens before the flow will continue to the next activity. Section 9.5.3, "Inclusive Gateways," on page 81 for more information on Inclusive Gateways.
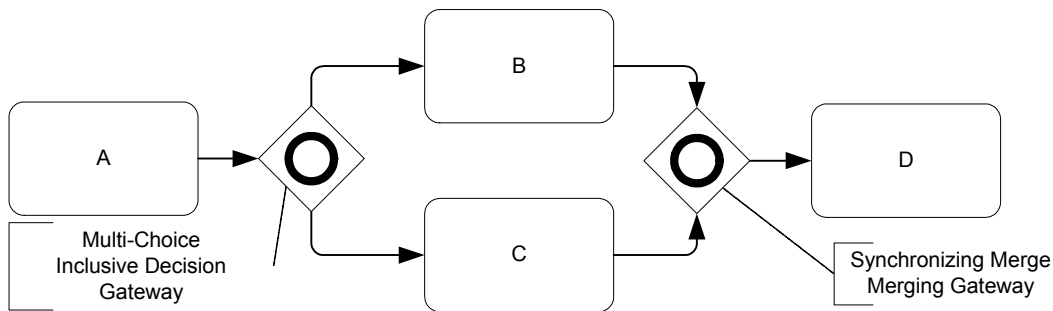
Figure 10.33 - Workflow Pattern #9 -- Synchronizing Join

The fourth type of Workflow Pattern merge is called a N out of M Join[12]. This type of situation is more complex and can be handled through a Complex Gateway (see Figure 10.34). The Gateway will receive Tokens from its incoming Sequence Flow and evaluate an expression to determine whether or not the flow should proceed. Once the condition has been satisfied, if additional Tokens arrive, they will be excluded (much like the Discriminator Pattern from Figure 10.32). Section 9.5.4, "Complex Gateways," on page 84 for more information on Complex Gateways.
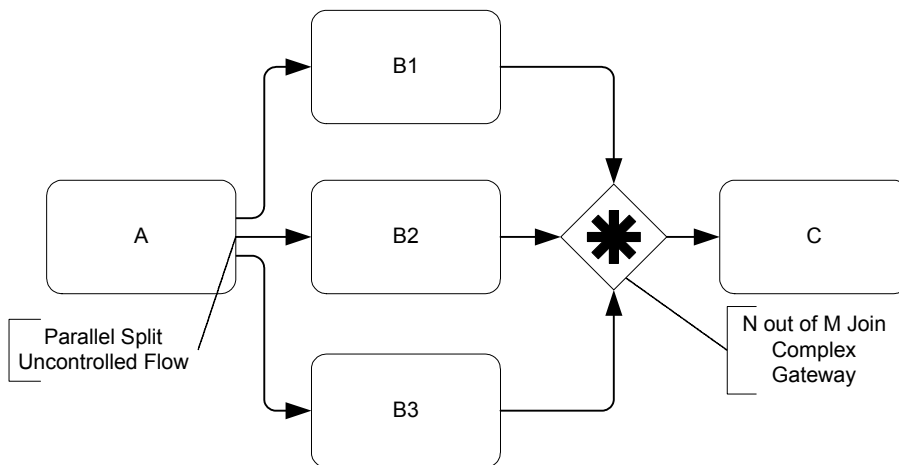
Figure 10.34 - Workflow Pattern #8 -- N out of M Join

There is no specific correlation between the merging of a set of paths and the splitting that occurs through a Gateway object. For example, a Decision may split a path into three separate paths, but these three paths do not need to be merged at the same object. Figure 10.35 shows that two of three alternative paths are merged at Task "F." All of the paths eventually will be merged, but this can happen through any combination of objects, including lone End Events. In fact, each path could end with a separate End Event.

---

12. http://tmitwww.tm.tue.nl/research/patterns/n_out_of_m_join.htm
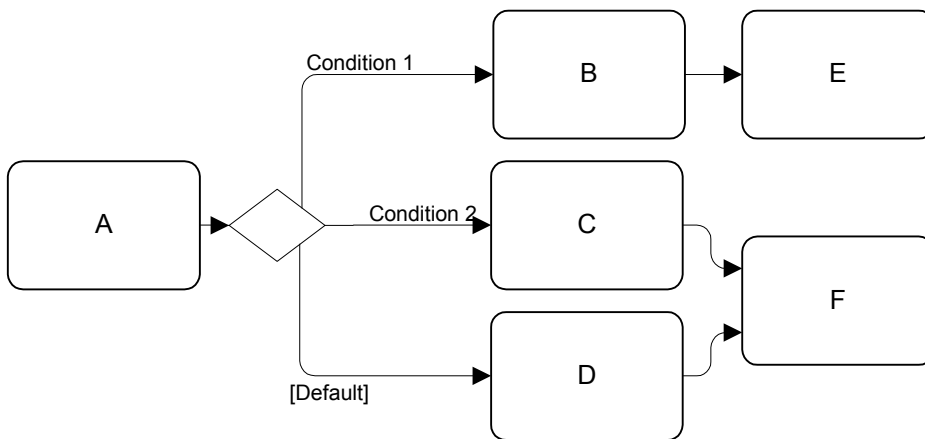
Figure 10.35 - The Split-Merge Relationship is not Fixed

Looping

BPMN provides 2 (two) mechanisms for looping within a Process. The first involves the use of attributes of activities to define the loop. The second involves the connection of Sequence Flow to "upstream" objects.

Activity Looping

The attributes of Tasks and Sub-Processes will determine if they are repeated as a loop. There are two types of loops that can be specified: Standard and Multi-Instance.

For Standard Loops:

- If the loop condition is evaluated before the activity, this is generally referred to as a "while" loop. This means that the activities will be repeated as long as the condition is true. The activities may not be performed at all (if the condition is false the first time) or performed many times.

- If the loop condition is evaluated after the activity, this is generally referred to as an "until" loop. This means that the activities will be repeated until a condition becomes true. The activities will be performed at least once, but may be performed many times.

For Multi-Instance Loops:

- If the MI_Ordering is serial, then this becomes much like a while loop with a set number of iterations the loop will go through. These are often used in processes where a specific type of item will have a set number of sub-items or line items. A Multi-Instance loop will be used to process each of the line items.

- If the MI_Ordering is parallel, this is generally referred to as multiple instances of the activities. An example of this type of feature would be used in a process to write a book, there would be a Sub-Process to write a chapter. There would be as many copies or instances of the Sub-Process as there are chapters in the book. All the instances could begin at the same time.

Those activities that are repeated (looped) will have a loop marker placed in the bottom center of the activity shape (see Figure 10.36). Those activities that are Parallel Multi-Instance will have a parallel marker placed in the bottom center of the activity shape (see Figure 10.37)
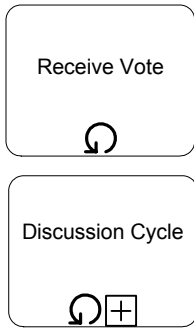
Figure 10.36 - A Task and a Collapsed Sub-Process with a Loop Marker
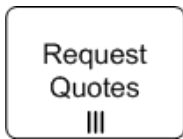
Figure 10.37 - A Task with a Parallel Marker

Expanded Sub-Processes also can have a loop marker placed at the bottom center of the Sub-Process rectangle (see Figure 10.38). The entire contents of the Sub-Process will be repeated as defined in the attributes.
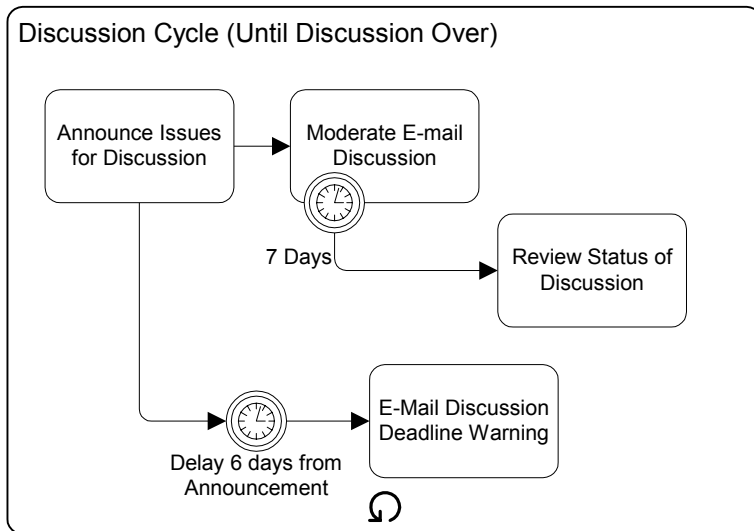
Figure 10.38 - An Expanded Sub-Process with a Loop Marker

Sequence Flow Looping

Loops can also 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 Flow, the last of which turns out to be an incoming Sequence Flow to the original object. That is, that object produces a Token and that Token traverses a set of Sequence Flow until the Token reaches the same object again. Sequence Flow looping is the same as Workflow Pattern #16 -- Arbitrary Cycle[13] (see Figure 10.24).
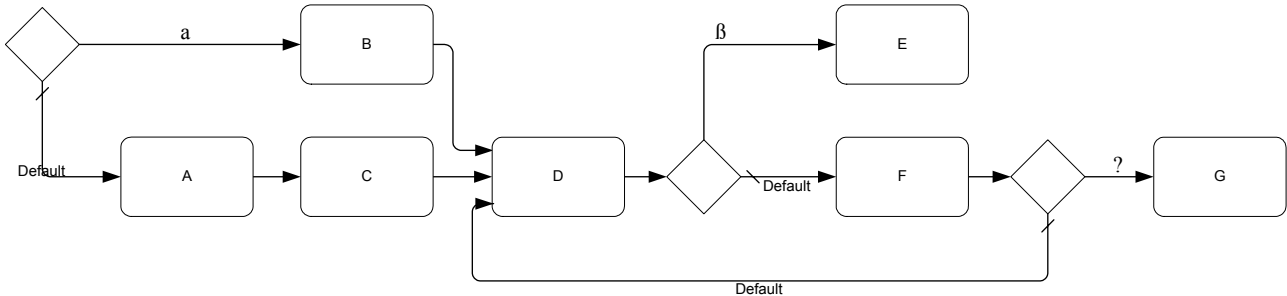
Figure 10.39 - Workflow Pattern #16 -- Arbitrary Cycle

Usually these connections follow a Decision so that the loop is not infinite (see Figure 10.40). If the Sequence Flow goes directly from a Decision to an upstream object, this is an "until" loop. The set of looped activities will occur until a certain condition is true.
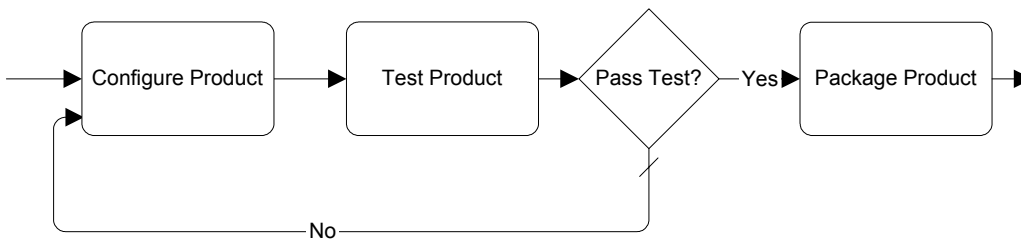
Figure 10.40 - An Until Loop

A while loop is created by making the decision first and then performing the repeating activities or moving on in the Process (see Figure 10.41). The set of looped activities may not occur or may occur many times.

---

13. http://tmitwww.tm.tue.nl/research/patterns/arbitrary_cycle.htm
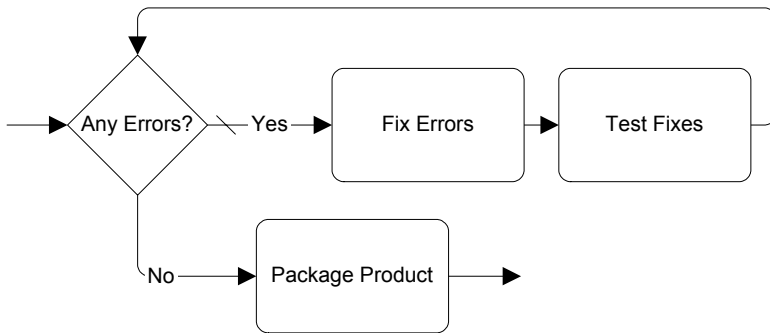
Figure 10.41 - A While Loop

Sequence Flow Jumping (Off-Page Connectors and Go To Objects)

Since process models often extend beyond the length of one printed page, there is often a concern about showing how Sequence Flow connections extend across the page breaks. One solution that is often employed is the use of Off-Page connectors to show where one page leaves off and the other begins. BPMN provides Intermediate Events of type Link for use as Off-Page connectors (see Figure 10.42--Note that the figure shows two different printed pages, not two Pools in one diagram). A pair of Link Intermediate Events is used. One of the pair is shown at the end of one page. This Event is named and has an incoming Sequence Flow and no outgoing Sequence Flow. The second Link Event is at the beginning of the next page, shares the same name, and has an outgoing Sequence Flow and no incoming Sequence Flow.
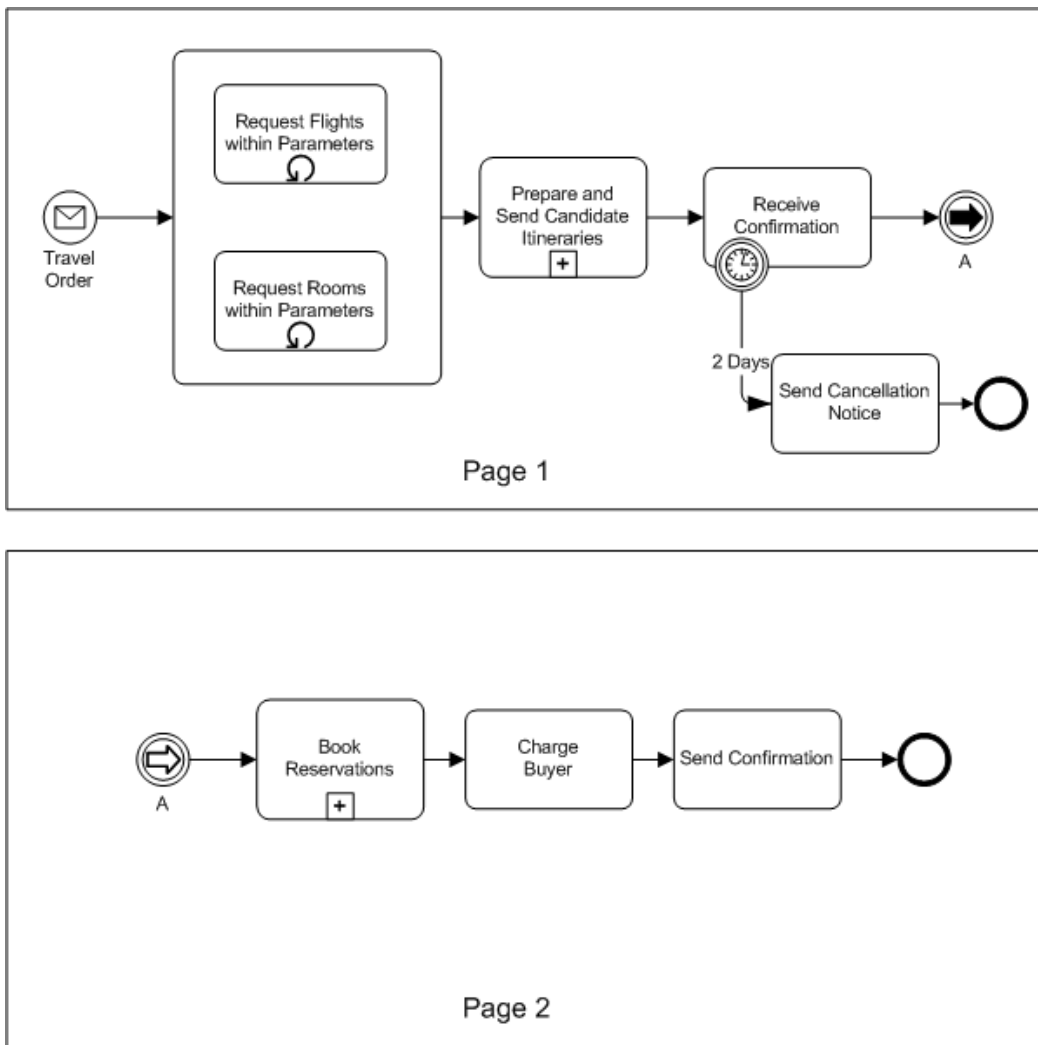
Figure 10.42 - Link Intermediate Event Used as Off-Page Connector

Another way that Link Intermediate Events can be used is as "Go To" objects. Functionally, they would work the same as for Off-Page Connectors (described above), except that they could be used anywhere in the diagram--on the same page or across multiple pages. The general idea is that they provide a mechanism for reducing the length of Sequence Flow lines. Some modelers may consider long lines as being hard to follow or trace. Go To Objects can be used to avoid very long Sequence Flow (see Figure 10.43 and Figure 10.44). Both diagrams will behave equivalently. For Figure 10.44, if the "Order Rejected" path is taken from the Decision, then the Token traversing the Sequence Flow would reach the source Link Event and then "jump" to the target Link Event and continue down the Sequence Flow. The process would continue as if the Sequence Flow had directly connected the two objects.
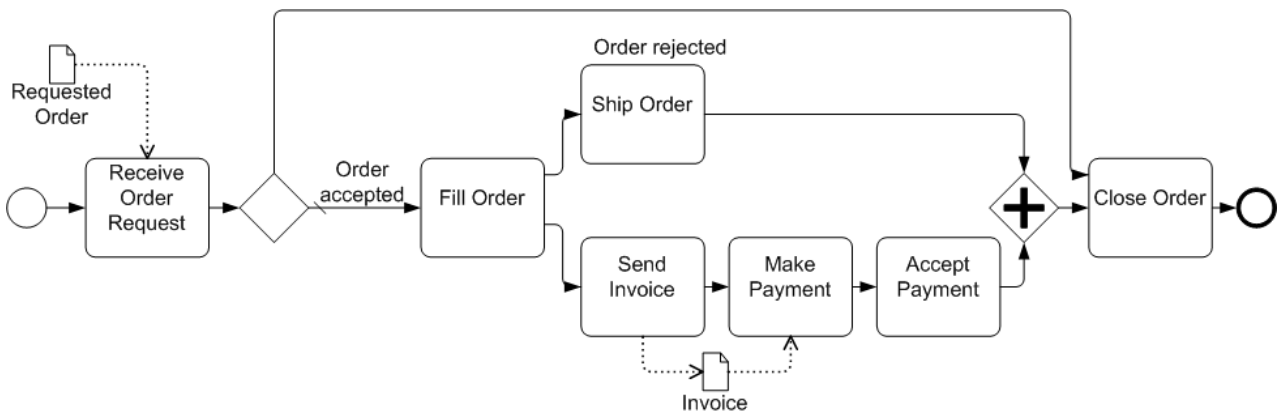
Figure 10.43 - Process with Long Sequence Flow
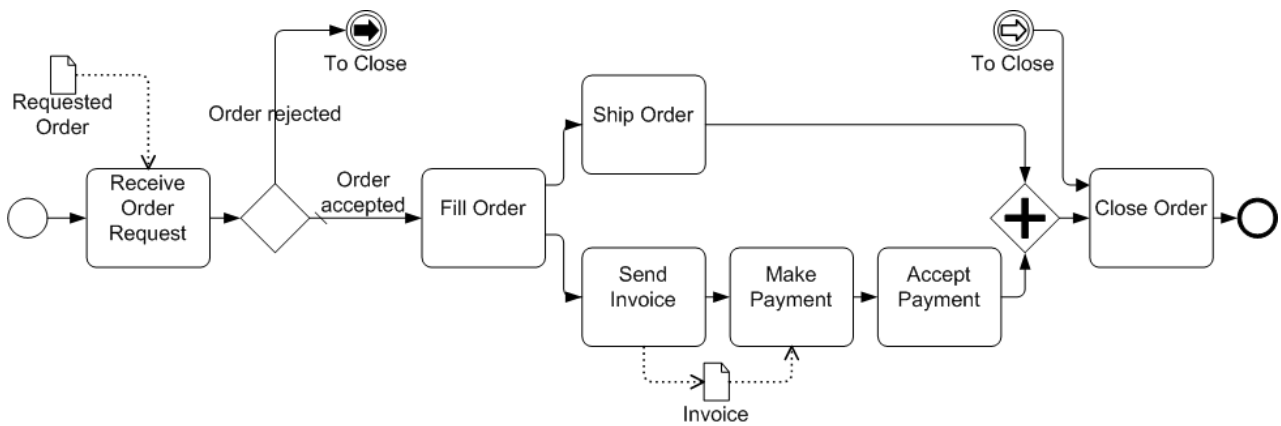
Figure 10.44 - Process with Link Intermediate Events Used as Go To Objects

Some methodologies prefer that all Sequence Flow only move in one direction; that is, forward in time. These methodologies do not allow Sequence Flow to connect directly to upstream objects. Some consistency in modeling can be gained by such a methodology, but situations that require looping become a challenge. Link Intermediate Events can be used to make upstream connections and create loops without violating the Sequence Flow direction restriction (see Figure 10.45).
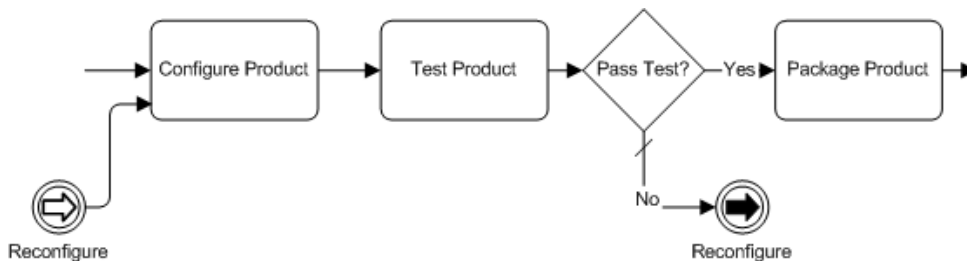
Figure 10.45 - Link Intermediate Event Used for Looping

Passing Flow to and from Sub-Processes

This section reviews how flow will be passed between a parent Process and any of its Sub-Processes. The flow (e.g., a Token) will start at the parent Process and then move to the Sub-Process and then will move back to the parent process (see Figure 10.46). Most of the time the flow will reach a Sub-Process, get transferred to the Start Event of the Sub-Process, traverse the Sequence Flow of the Sub-Process, reach the End Event of the Sub-Process, and, finally, get transferred back to the parent Process to continue down the outgoing Sequence Flow of the Sub-Process object. If the Sub-Process contains parallel Flow, then all the Flow must complete before a Token is transferred back to the parent Process. This functionality treats the Sub-Process as a self-contained "box" of activities.
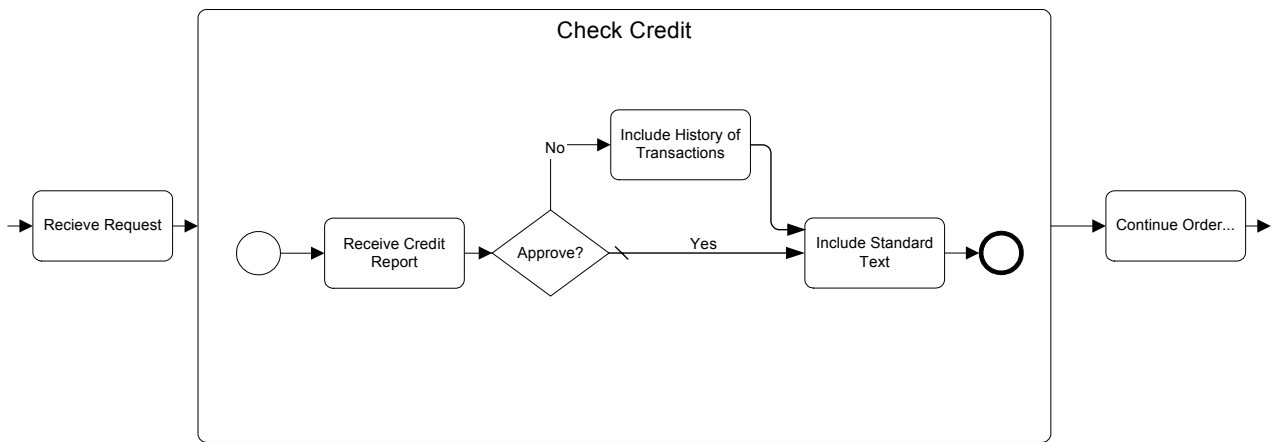
Figure 10.46 - Example of Sub-Process with Start and End Events Inside

To make the flow between levels of a Process more obvious, a modeler has the option of placing the Start Event and the End Event on the boundary of the Sub-Process and connect the Sequence Flow from the Parent Process objects to/from these Events (see Figure 10.47).
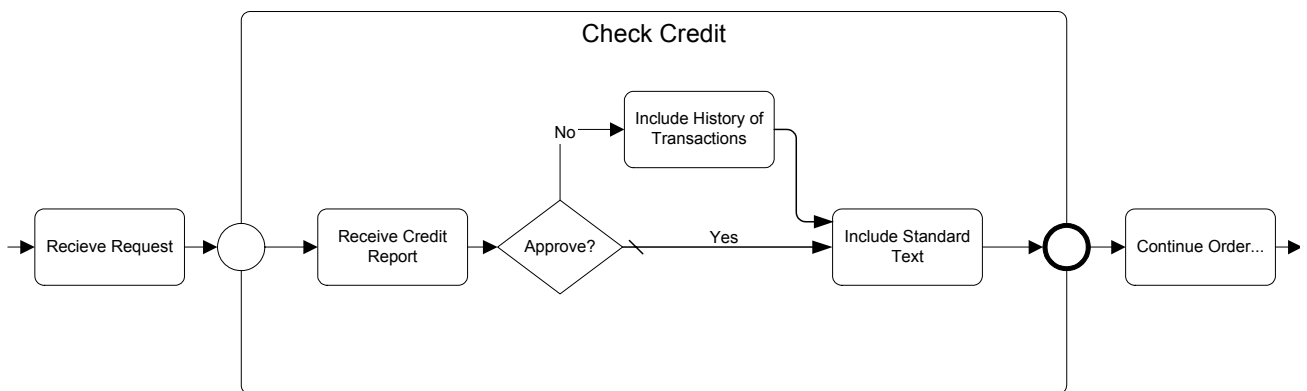
Figure 10.47 - Example of Sub-Process with Start and End Events on Boundary

Controlling Flow Across Processes

There may be situations within a Process where the flow is affected by or dependent on the activity that occurs in another Process. These events or conditions can be referred to as milestones. The process model must be able to identify and react to the milestone.That is, the continuation of a Process may be triggered by Signal Events, which pass the flow between processes (see Figure 10.48). The type of Workflow Pattern called a Milestone[14].
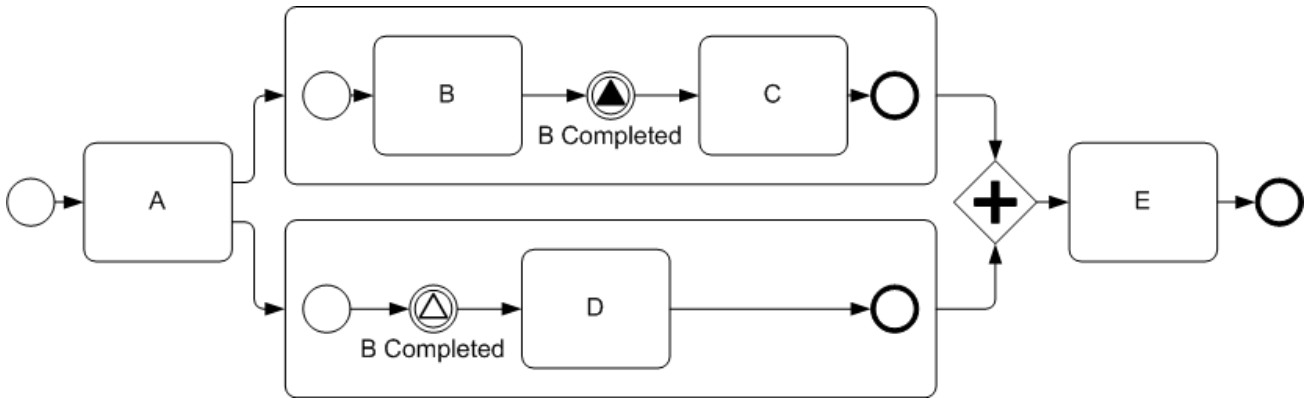
Figure 10.48 - Signal Events Used to Synchronize Behavior Across Processes

Avoiding Illegal Models and Unexpected Behavior

BPMN, being a graph-structured Diagram, rather than having a block-structures like BPEL4WS, provides a great flexibility for depicting complex process behavior in a fairly compact form. However, the free-form nature of BPMN can create modeling situations that cannot be executed or will behave in a manner that is not expected by the modeler. These types of modeling problems can occur because there is not a tight relationship between forks and joins or splits and merges. A block structure provides these tight relationships, but a graph-structure allows these flow control mechanisms to be mixed and matched at the discretion of the modeler. Some combinations of these control elements will create Processes that cannot be executed or will create behavior that was not intended by the modeler. The situation where alternative paths cross the implicit boundary of a group of parallel paths can cause an invalid model.

Figure 10.49 shows such a model. Task "D" is an activity that has two incoming Sequence Flow; one from a forked path (after a split path) and one from a split path. This can create a problem at the Parallel Gateway that precedes Task "E," which also has multiple incoming Sequence Flow. The Sequence Flow from Task "B" is crossing the implicit boundary of the fork created after Task "A." As a result, if the "Yes" Sequence Flow is taken from the Decision in the Diagram (Variation 1), then Task "E" can expect two Tokens to arrive—one from Task "C" and one from Task "D." However, if the "No" Sequence Flow is taken from the Decision (Variation 2), the Parallel Gateway will receive only one Token—one from Task "D." Since the Gateway expects two Tokens, the Process will be dead-locked at that position.

14. http://tmitwww.tm.tue.nl/research/patterns/milestone.htm
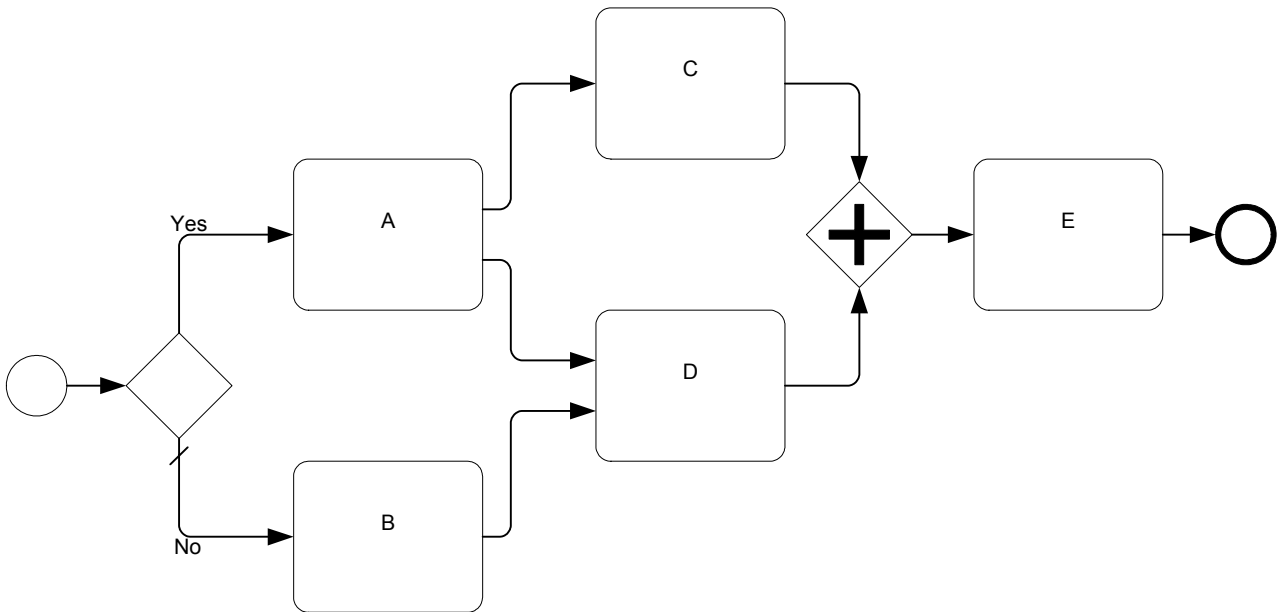
Figure 10.49 - Potentially a dead-locked model

Another type of problem occurs with looping back to upstream activities. If the loop Decision is made within the implicit boundaries of a set of parallel paths, then the behavior of the loop becomes ambiguous (see Figure 10.50), since it is unclear whether Task "E" was intended to be repeated based on the loop or what would happen if Task "E" was still active when the loop reached that Task again.
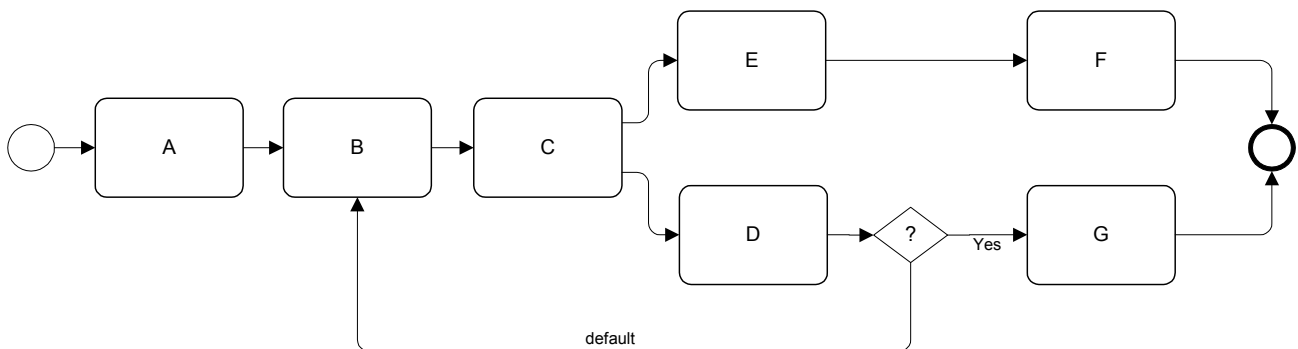
Figure 10.50 - Improper Looping

The use of Link Events can also create unexpected behavior. In general, Link Events not used for off-page connectors should be considered an advanced modeling technique and the modeler should be careful to understand the resultant behavior and flow of Tokens.

In general, the analysis of how Tokens will flow through the model will help find models that cannot be executed properly. This Token flow analysis will be used to create some of the mappings to BPEL4WS. Since BPEL4WS is properly executable, if the Token flow analysis cannot create a valid BPEL4WS process, then the model is not structured correctly.

10.2.2 Exception Flow

Exception Flow occurs outside the Normal Flow of the Process and is based upon an event (an Intermediate Event) that occurs during the performance of the Process. While Intermediate Events can be included in the Normal Flow to set delays or breaks to wait for a message, when they are attached to the boundary of an activity, either a Task or a Sub-Process (see Figure 10.51), they create Exception Flow.
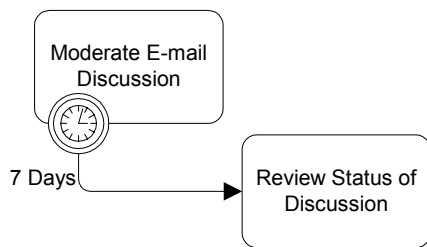
Figure 10.51 - A Task with Exception Flow (Interrupts Event Context)

By doing this, the modeler is creating an Event Context. The Event Context will respond to specific Triggers to interrupt the activity and redirect the flow through the Intermediate Event. The Event Context will only respond if it is active (running) at the time of the Trigger. If the activity has completed, then the Trigger may occur with no response. The source of the Trigger may be external to the Process execution, such as a message or an application error, or the Trigger may be caused by a "throw" Intermediate Event from any other active location within the Process. An exception to this is the Error event, which will only respond to Error triggers generated within the activity or in a subprocess of that activity.

If there are a group of Tasks that the modeler wants to include in an Event Context, then a Sub-Process can be added to encompass the Tasks and to handle any events by having them attached to its boundary (see Figure 10.52).
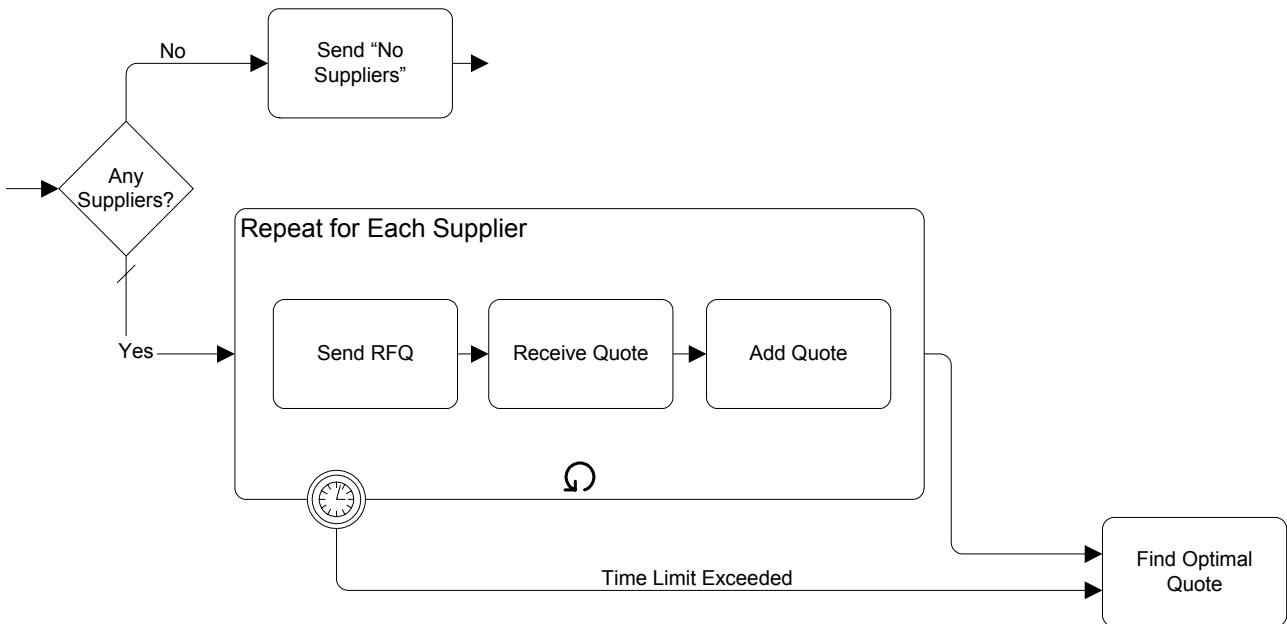
Figure 10.52 - A Sub-Process with Exception Flow (Interrupts Event Context)

A Message Event occurs when a message, with the exact identity as specified in the Intermediate Event, is received by the Process. An Error Event occurs when the Process detects an Error. If an Error Code is specified in the Intermediate Event, then the code of the detected Error must match for the Event Context to respond. If the Intermediate Event does not specify an Error Code, then any Error will trigger a response from the Event Context.If this event does not occur while the Event Context is ready, then the Process will continue through the Normal Flow as defined through the Sequence Flow.

10.2.3 Ad Hoc

An Ad Hoc Process is a group of activities that have no pre-definable sequence relationships. A set of activities can be defined for the Process, but the sequence and number of performances for the activities is completely determined by the performers of the activities and cannot be defined beforehand.

A Sub-Process is marked as being an Ad Hoc with a "tilde" symbol placed at the bottom center of the Sub-Process shape (see Figure 10.53 and Figure 10.54). Activities within the Process are disconnected from each other. During execution of the Process, any one or more of the activities may be active and they can be performed in almost any order or frequency.
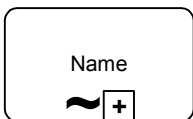
Figure 10.53 - A Collapsed Ad Hoc Sub-Process
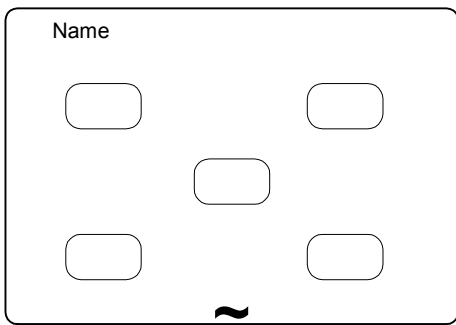
Figure 10.54 - An Expanded Ad Hoc Sub-Process

The performers determine when activities will start, when they will end, what the next activity will be, and so on. Examples of the types of Processes that are Ad Hoc include computer code development (at a low level), sales support, and writing a book chapter. If we look at the details of writing a book chapter, we could see that the activities within this Process include: researching the topic, writing text, editing text, generating graphics, including graphics in the text, organizing references, etc. (see Figure 10.55). There may be some dependencies between Tasks in this Process, such as writing text before editing text, but there is not necessarily any correlation between an instance of writing text to an instance of editing text. Editing may occur infrequently and based on the text of many instances of the writing text Task.
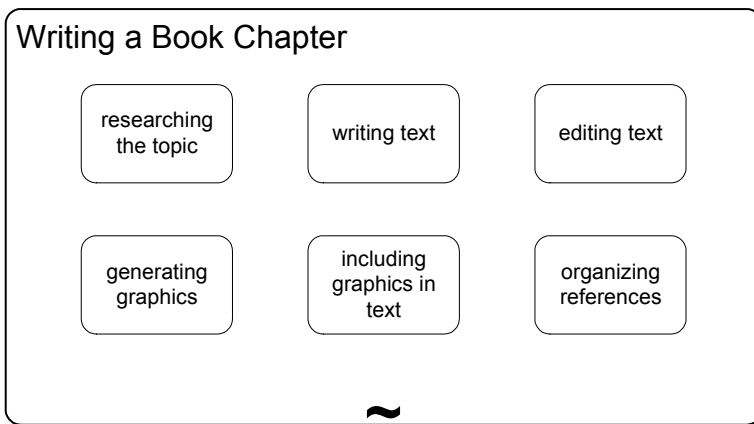
Figure 10.55 - An Ad Hoc Process for Writing a Book Chapter

It is a challenge for a BPM engine to monitor the status of Ad Hoc Processes, usually these kind of processes are handled through groupware applications (such as e-mail), but BPMN allows modeling of Processes that are not necessarily executable and should provide the mechanisms for those BPM engines that can follow an Ad Hoc Process. Given this, at some point, the Process will have completed and this can be determined by evaluating a Completion Condition that evaluates Process attributes that will have been updated by an activity in the Process.

10.3 Compensation Association

Some activities produce complex effects or specific outputs. If the outcome is determined to be undesirable by some specified criteria (such as an order being cancelled), then it will be necessary to "undo" the activities. There are three ways this can be done:

- Restoring of a copy of the initial values for data, thereby overwriting any changes.

- Doing nothing (if nothing has be changed because the changes have been set aside until a confirmation).

- Invoking activities that undo the effects--also known as compensation.

An activity that might require compensation could be, for example, one that charges a buyer for some service and debits a credit card to do so. These types of activities usually need a separate activity to counter the effects of the initial activity. Often, a record of both activities is required, so this is another reason that the activity is not "undone." An Intermediate Event of type Compensation is attached to the boundary of an activity to indicate that compensation may be necessary for that activity.

One of the three mechanisms for "undo" activities, Compensation, requires specific notation and is a special circumstance that occurs outside the Normal Flow of the Process. For this reason, the Compensation Intermediate Event does not have an outgoing Sequence Flow, but instead has an outgoing directed Association (see Figure 10.56).
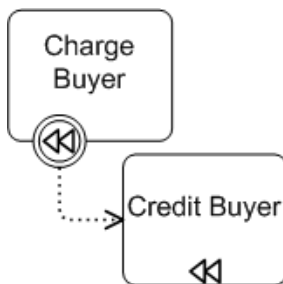
Figure 10.56 - A Task with an Associated Compensation Activity

The target of this Association is the activity that will compensate for the work done in the source activity, and will be referred to as the Compensation Activity. The Compensation Activity is special in that it does not follow the normal Sequence Flow rules--as mentioned, it is outside the Normal Flow of the Process. This activity cannot have any incoming or outgoing Sequence Flow. The Compensation marker (as is in the Compensation Intermediate Event) will be displayed in the bottom center of the Activity to show this status of the activity (see the "Credit Buyer" Task in Figure 10.56). Note that there can be only one target activity for compensation. There cannot be a sequence of activities shown. If the compensation does require more than one activity, then these activities must be put inside a single Sub-Process that is the target of the Association. The Sub-Process can be collapsed or expanded. If the Sub-Process is expanded, then only the Sub-Process itself requires the Compensation marker--the activities inside the Sub-Process do not require this marker.

Only activities that have been completed can be compensated. The compensation of an activity can be triggered in two ways:

- The activity is inside a Transaction Sub-Process that is cancelled (see Figure 10.57). In this situation, the whole Sub-Process will be "rewound" or rolled back--the Process flow will go backwards and any activity that requires compensation will be compensated. This is why the Compensation marker for Events looks like a "rewind" symbol for a tape player. After the compensation has been completed, the Process will continue its rollback.

- A downstream Intermediate or End Event of type Compensation "throws" a compensation identifier that is "caught" by the Intermediate Event attached to the boundary of the activity. The compensation is thrown in two ways:

  - The Event can specifically identify an activity that requires compensation and only that activity will be compensated.

  - The Event can broadcast the need for the compensation and then all completed activities that have a Compensation

Intermediate Event attached to their boundaries will be compensated. The compensation applies to all activities that have fully completed within the Process Instance (which includes all levels of the Process). The compensation will occur in the reverse order of the original performances on the triggered activities.
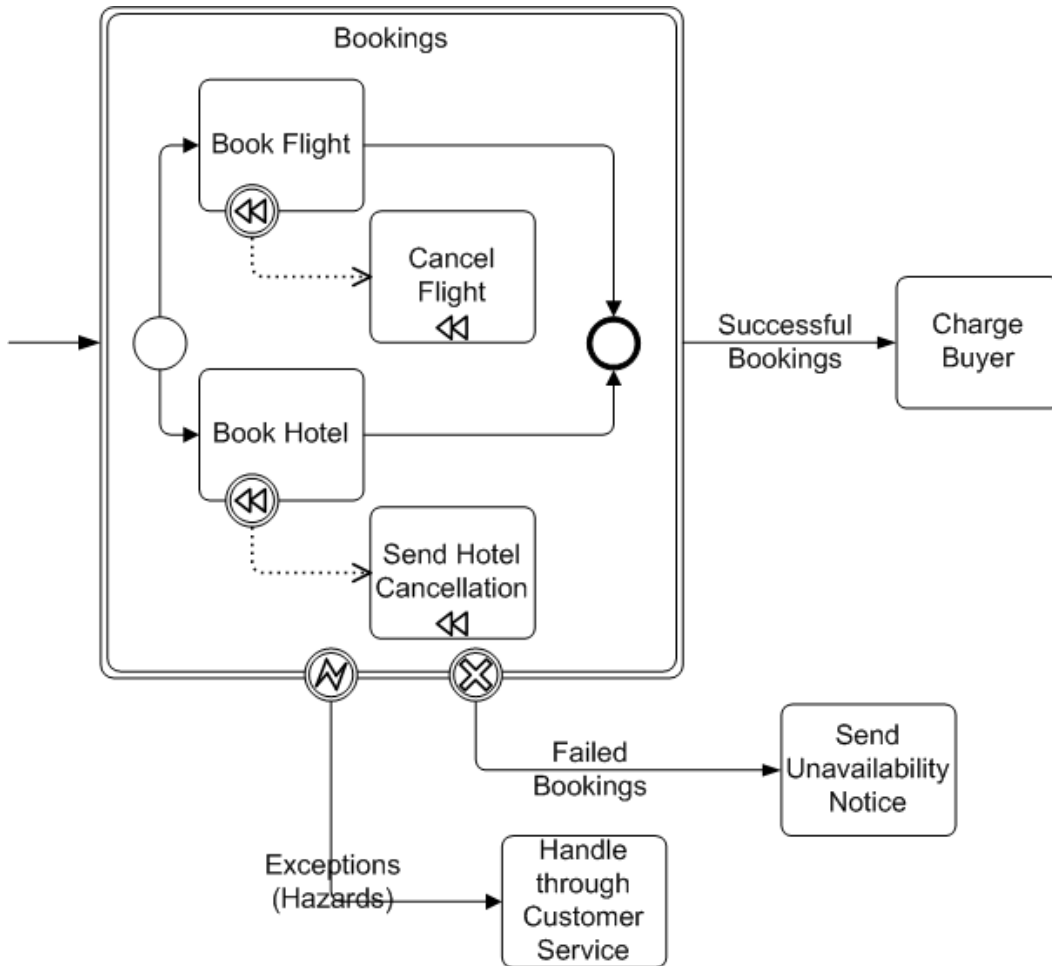
Figure 10.57 - Compensation Shown in the context of a Transaction

11 BPMN by Example

This section will provide an example of a business process modeled with BPMN. The process that will be described is a process used to help develop this notation. It is a process for resolving issues through e-mail votes (see Figure 11.1). This Process is small, but fairly complex and will provide examples for many of the features of BPMN. There are some unusual features of this business process, such as infinite loops. Although not a typical process, it will help illustrate that BPMN can handle simple and unusual business processes and still be easily understandable for readers of the Diagram. The sections below will isolate segments of the Process and highlight the modeling features as the workings of the Process is described.
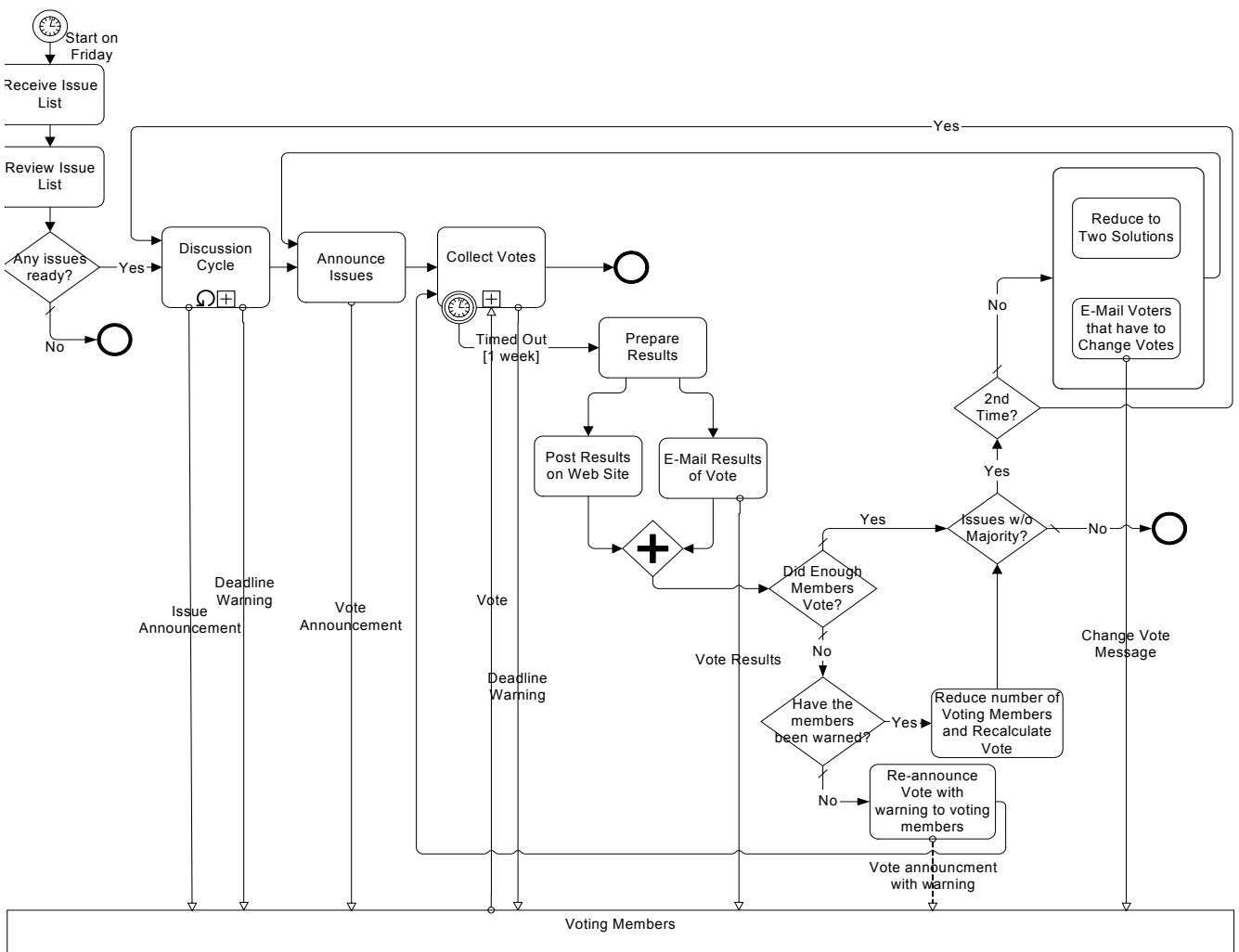
Figure 11.1 - E-Mail Voting Process

The Process has a point of view that is from the perspective of the manager of the Issues List and the discussion around this list. From that point of view, the voting members of the working group are considered as external Participants who will be communicated with by messages (shown as Message Flow).

11.1 The Beginning of the Process

The Process starts with Timer Start Event that is set to trigger the Process every Friday (see Figure 11.2).
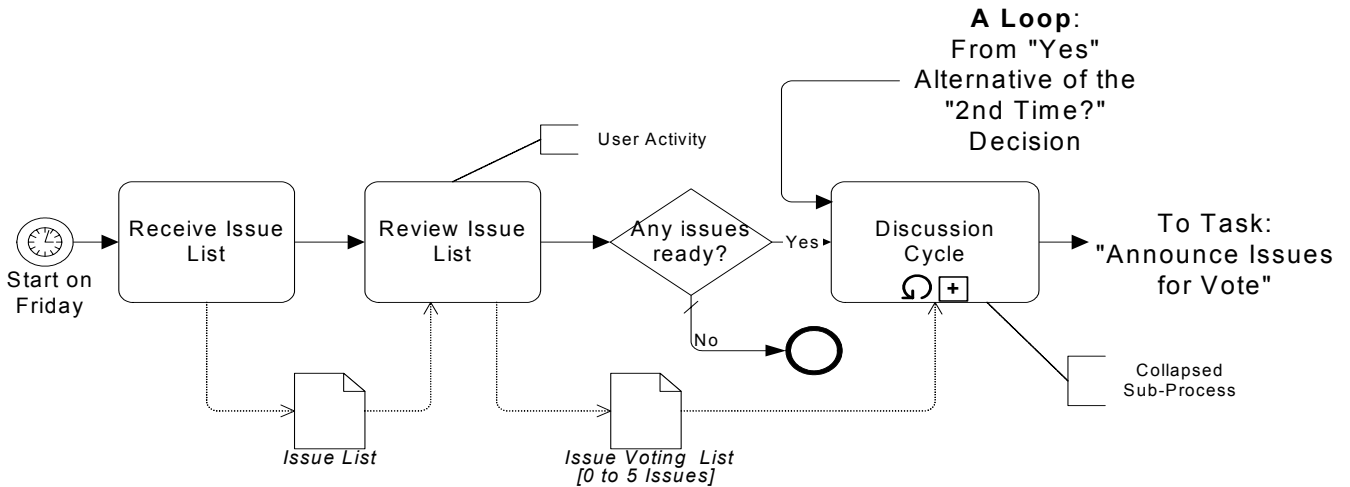
Figure 11.2 - The Start of the Process

The Issue List Manager will review the list and determine if there are any issues that are ready for going through the discussion and voting cycle. Then a Decision must be made. If there are no issues ready, then the Process is over for that week--to be taken up again the following week. If there are issues ready, then the Process will continue with the discussion cycle. The "Discussion Cycle" Sub-Process is the first activity after the "Any issues ready?" Decision and this Sub-Process has two incoming Sequence Flow, one of which originates from a downstream Decision and is thus part of a loop. It is one of a set of five complex loops that exist in the Process. The contents of the "Discussion Cycle" Sub-Process and the activities that follow will be described below.

11.2 The First Sub-Process

Figure 11.3 shows the details of the "Discussion Cycle" as an Expanded Sub-Process.
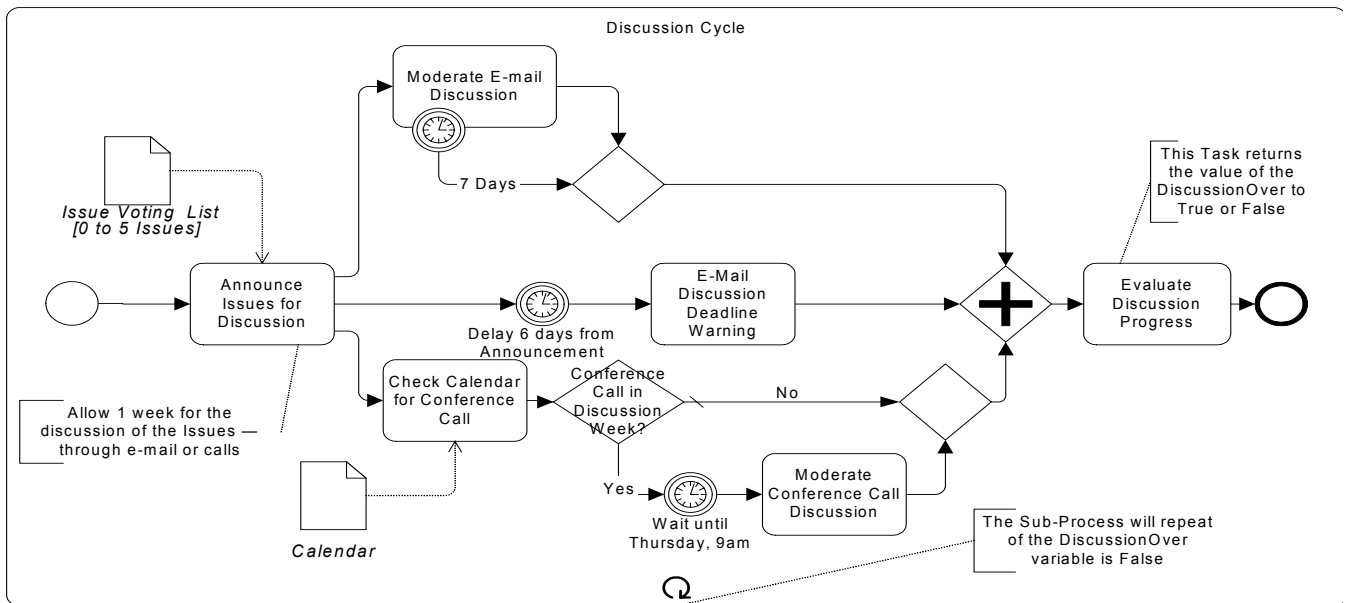
Figure 11.3 - "Discussion Cycle" Sub-Process Details

The Sub-Process starts of with a Task for the Issue List Manager to send an e-mail to the working group that a set of Issues are now open for discussion through the working group's message board. Since this Task sends a message to an outside Participant (the working group members), an outgoing Message Flow is seen from the "Discussion Cycle" Sub-Process to the "Voting Members" Pool in Figure 11.1. Basically, the working group will be discussing the issues for one week and proposing additional solutions to the issues. After the first Task, three separate parallel paths are followed, which are synchronized downstream. This is shown by the three outgoing Sequence Flow for that activity.

The top parallel path in the figure starts with a long-running Task, "Moderate E-mail Discussion," that has a Timer Intermediate Event attached to its boundary. Although the "Moderate E-Mail Discussion" Task will never actually be completed normally in this model, there must be an outgoing Sequence Flow for the Task since Start and End Events are being used within the Process. This Sequence Flow will merged with the Sequence Flow that comes from the Timer Intermediate Event. A merging Exclusive Gateway is used in this situation because the next object is a joining Parallel Gateway (the diamond with the cross in the center) that is used to synchronize the three parallel paths. If the merging Gateway was not used and both Sequence Flow connected to the joining Gateway, the Process would have been stuck at the joining Gateway that would wait for a Token to arrive from each of the incoming Sequence Flow.

The middle parallel path of the fork contains an Intermediate Event and a Task. A Timer Intermediate Event used in the middle of the Process flow (not attached to the boundary of an activity) will cause a delay. This delay is set to 6 days. The "E-Mail Discussion Deadline Warning" Task will follow. Again, since this Task sends a message to an outside Participant, an outgoing Message Flow is seen from the "Discussion Cycle" Sub-Process to the "Voting Members" Pool in Figure 11.1.

The bottom parallel path of the fork contains more than one object, first of which is Task where the issue list manager checks the calendar to see if there is a conference call this week. The output of the Task will be an update to the variable "ConCall," which will be true or false. After the Task, an Exclusive Gateway with its two Gates follows. The Gate for labeled "default" Flow directly to an merging Exclusive Gateway, for the same reason as in the top parallel path. The Gate for the "Yes" Sequence Flow will have a *condition* that checks the value of the "ConCall" variable (set in the

previous Task) to see if there will be a conference call during the coming week. If so, the Timer Intermediate Event indicates delay, since all conference calls for the working group start at 9am PDT on Thursdays. The Task for moderating the conference call follows the delay, which is followed the merging Gateway.

The merging Gateways in the top and bottom paths and the "E-Mail Discussion Deadline Warning" Task all flow into a joining Gateway. This Gateway waits for all three paths to complete before the Process Flow to the next Task, "Evaluate Discussion Progress." The issue list manager will review the status of the issues and the discussions during the past week and decide if the discussions are over. The DiscussionOver variable will be set to TRUE or FALSE, depending on this evaluation. If the variable is set to FALSE, then the whole Sub-Process will be repeated, since it has looping set and the loop condition will test the DiscussionOver variable.

11.3 The Second Sub-Process

Figure 11.4 shows the next section of the Process, which includes the expanded details of the "Collect Votes" Sub-Process.
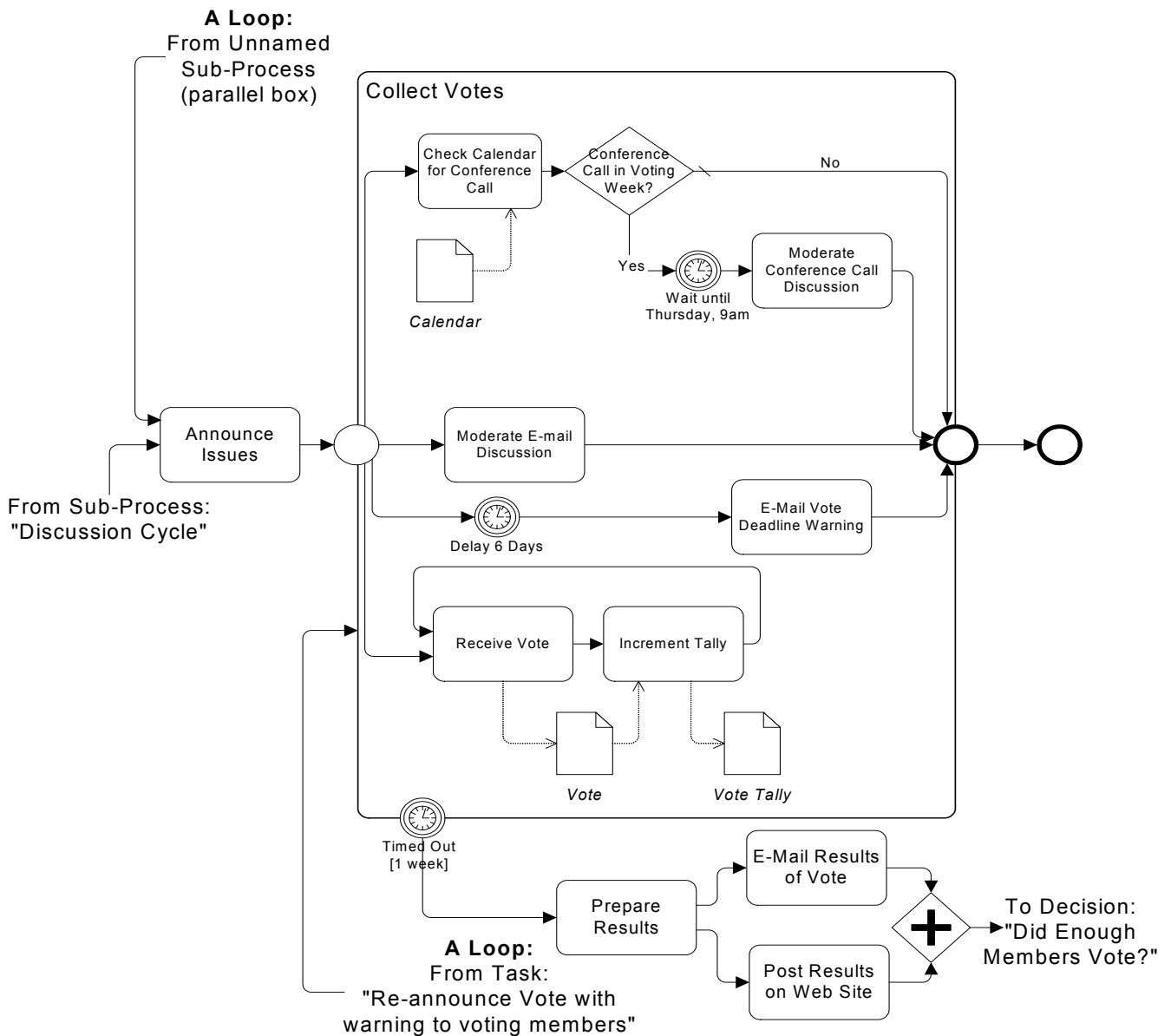
Figure 11.4 - "Collect Votes" Sub-Process Details

This part of the process starts out with a Task for the issue list manager to send out an e-mail to announce to the working group, and the voting members in particular, which lets them know that the issues are now ready for voting. Since this Task sends a message to an outside Participant (the working group members), an outgoing Message Flow is seen from the "Announce Issues" Task to the "Voting Members" Pool in Figure 11.1. This Task is also a target for one of the complex loops in the Process.

The "Collect Votes" Sub-Process follows the Task, and is also a target of one of the looping Sequence Flow. This Sub-Process is basically a set of four parallel paths that extend from the beginning to the end of the Sub-Process.

The first branch of the fork leads to a Decision that determines whether or not a conference call will occur during the upcoming week, after the Working Group's schedule has been checked. Basically, if there was a call last week, then there will not be a call this week and vice versa. The appropriate variable that was updated in the "Discussion Cycle" Process will be used again.

The second and third branches forks work the same way as the similar activities in the "Discussion Cycle" Sub-Process, except that the "Moderate E-Mail Discussion" Task does not have a Timer Intermediate Event attached. This is not necessary since the whole Sub-Process is interrupted after 7 days through the Intermediate Event attached to the Sub-Process boundary. The "E-Mail Vote Deadline Warning" Task sends a message to an outside Participant (the working group members), thus, an outgoing Message Flow is seen from the "Collect Votes" Sub-Process to the "Voting Members" Pool in Figure 11.1.

The fourth branch of the fork is rather unique in that the Diagram uses a loop that does not utilize a Decision. Thus, it is, as it is intended to be, an infinite loop. The policy of the working group is that voting members can vote more than once on an issue; that is, they can change their mind as many times as they want throughout the entire week. The first Task in the loop receives a message from the outside Participant (the working group members), thus, an incoming Message Flow is seen from the "Voting Members" Pool to the "Collect Votes" Sub-Process in Figure 11.1. The Timer Intermediate Event attached to the boundary of the Sub-Process is the mechanism that will end the infinite loop, since all work inside the Sub-Process will be ended when the time-out is triggered. All the remaining work of the Process is conducted after the time-out and Flow from the Timer Intermediate Event.

Figure 11.4 shows that there are Two Tasks that follow the time-out. First, a Task will prepare all the voting results, then a Task will send the results to the voting members. A Document Object, "Issue Votes," is shown in the Diagram to illustrate how one might be used, but it will not map to anything in the execution languages. The remaining activities of the Process will be described in the next section.

11.4   The End of the Process

Figure 11.5 shows the last section of the Process, which includes a complex set of Decisions and loops.
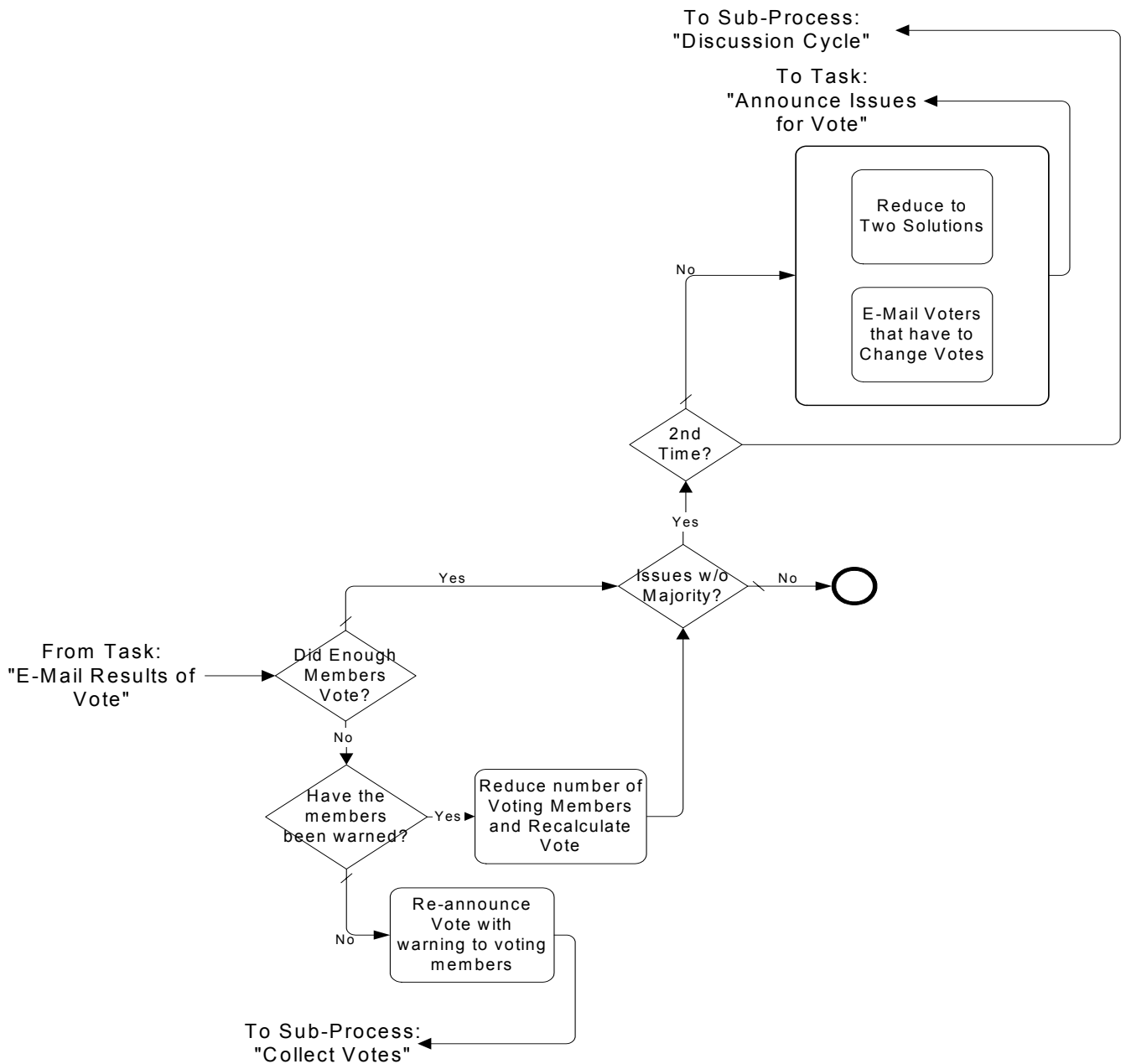
Figure 11.5 - The last segment of the E-Mail Voting Process

This segment of the Process continues from where the last segment left off (as described in the section above). It contains four Decisions that interact with each other and create loops to upstream activities.

The first Decision, "Did Enough Members Vote?," is necessary since two-thirds of the voting members are required to approve any solution to an issue. If less than two-thirds of the voting members cast votes, which sometimes happens, the issues can't be resolved. This Decision Flow to another Decision for both of its Alternatives. The "No" Alternative is followed by the "Have the Members been Warned?" Decision. If a voting member misses a vote, they are warned. If they

miss a second vote, they lose their status as a voting member and the voting percentages are recalculate through a Task ("Reduce number of Voting Members and Recalculate Vote"). If they haven't yet been warned, then a warning is sent and the voting week is repeated.

If all issues are resolved, then the Process is done. If not, then another Decision is required. The voting is given two chances before it goes back to another cycle of discussion. The first time will see a reduction of the number of solutions to the two most popular based on the vote (more if there are ties). Some voting members will have to change their votes just because their solution is no longer valid. These two activities are placed in a Sub-Process to show how a Sub-Process without Start and End Events can be used to create a simple set of parallel activities. Informally, this is called a "parallel box." It is not a special object, but another use of Sub-Processes. For simple situations, it can be used to show a set of parallel activities without the extra clutter of a lot of Sequence Flow. In actuality, these two Tasks cannot actually be done in parallel, but they are modeled this way to highlight the optional use of Start and End Events.

After the parallel box, the flow loops back to the "Collect Votes" Sub-Process. If there already has been two cycles of voting, then the process Flow back to the "Decision Cycle" Sub-Process.

Annex A
(informative)

Mapping to BPEL4WS

This annex provides information and examples about how BPMN will map to BPEL4WS 1.1.

A.1 Mapping BPMN Elements and Diagrams to BPEL4WS

This section will cover a mapping to BPEL4WS that are derived by analyzing the BPMN objects and the relationships between these objects as described in the previous chapters. Note that there are known issues with the mapping as specified. Fixes to these issues will be incorporated in a later revision of the specification.

A.1.1 Business Process Diagram Mappings

A Business Process Diagram can be made up of a set of (semi-) independent components, which are shown as separate Pools. Thus, there is not a specific mapping to the diagram itself. Rather, there are separate mappings to each of the Pools that are in the diagram. That is, each Pool in the diagram, if it is a "white box" that contains process elements, will map to an individual BPEL4WS *process*. However, in the course of mapping the contents of the Process, there may be one or more derived *processes* necessary to handle complex behavior, such as looping. The attributes of "black box" Pools will also be used in determining specific BPEL4WS elements, such as *partnerLink*.

The following table displays a set of mappings for the attributes of a Business Process Diagram that can be mapped to BPEL4WS:

Table A.1 - Business Process Diagram Mappings to BPEL4WS

Business Process Diagram	Mapping to BPEL4WS
Id, Name, Version, Author, Language, CreationDate, ModificationDate, Pool, and Documentation	These Elements do not map to any BPEL4WS elements or attributes.
ExpressionLanguage attribute	This attribute will be used for all the Processes that are within the Business Process Diagram. The attribute will map to the *expressionLanguage* attribute of each BPEL4WS *process*.
QueryLanguage attribute	This attribute will be used for all the Processes that are within the Business Process Diagram. The attribute will map to the *queryLanguage* attribute of each BPEL4WS *process*.

A.1.2 Business Process Mappings

There can be one or more Business Processes within a Business Process Diagram, each within a separate Pool. The following table displays a set of mappings from attributes of a Process to BPEL4WS elements (the mappings for the objects contained within a Process, its contents, are mapped separately and these mappings can be found in the sections that follow):

Table A.2 - Business Process Mappings to BPEL4WS

Process	Mapping to BPEL4WS
ProcessType	If the Process is to be used to create a BPEL4WS document, then the attribute MUST be set to Private or Abstract. If the attribute is set to Private, then the *abstractProcess* attribute of the BPEL4WS *process* MUST be set to "no." If the attribute is set to Abstract, then the *abstractProcess* attribute of the BPEL4WS *process* MUST be set to "yes."
Id, Categories, and Documentation	These Elements do not map to any BPEL4WS elements or attributes.
Name	The Name attribute of the Process SHALL map to *name* attribute of the appropriate *process*. The extra spaces and non-alphanumeric characters MUST be stripped from the Name to fit with the XML specification of the *name* attribute. Note that there may be two or more elements with the same name after the BPMN name has been stripped.
GraphicalElements	This is a list of all the graphical elements contained within the Process. Each of these elements will have their mapping, as defined in the sections below.

Properties	The set of Properties of a Process, as a whole, will map to a BPEL4WS *variable*. The *variable* element will be structured as follows:
	
	`<variable name="[Process.Name]_Data`
	`        messageType="[Process.Name]_ProcessDataMessage" />`
	
	The individual Properties will map to the *parts* of a WSDL *message*. The *message* element will be structured as follows:
	
	`<message name="[Process.Name]_ProcessDataMessage" >`
	`  <part name="[Property.Name]"`
	`        type="xsd:[Property.Type]" />`
	`</message>`
	
	There will be as many *parts* to the *message* as there are Properties in the input group.

Correlation = True	This only applies to Properties of Type = "Set." The Name of the Property will map to the name of a *correlationSet*. The Name of each child Property for the Set will be added to the list of *properties* of the *correlationSet*.
Adhoc	Ad Hoc Processes are not executable. Thus, this attribute MUST be set to False if the Process is to be mapped to BPEL4WS.
AdHocCompletionCondition	This attribute only applies to Ad Hoc Processes. Thus, it will not be mapped to BPEL4WS.

Table A.2 - Business Process Mappings to BPEL4WS

Process	Mapping to BPEL4WS
With Assignments Expression	This will map to a BPEL4WS *assign*. Refer to the section entitled "Assignment Mapping" on page 195 for more details about the mappings associated with the *assign* element.
AssignTime = Start	A BPEL4WS *sequence* will be created and the *assign* will follow the instantiation of the process (through a *receive* or a *pick*).
AssignTime = End	A BPEL4WS sequence will be created and the *assign* will follow
SuppressJoinFailure	This maps to the BPEL4WS *process* attribute *suppressJoinFailure*.
EnableInstanceCompensation	This maps to the BPEL4WS *process* attribute *enableInstanceCompensation*.

   u   The BPEL4WS *process* attributes *targetNamespace* and *xmlns* MUST be provided by the modeling tool that generates the mapping to BPEL4WS.

A.1.3   Common Flow Object Mappings

The following table displays a set of mappings for the attributes common to Flow Objects (Events, Activities, and Gateways):

Table A.3 - Common Flow Object Attribute Mappings to BPEL4WS

Objects	Mapping to BPEL4WS
Id, Pool, Lanes, Categories, and Documentation	These Elements do not map to any BPEL4WS elements or attributes.
Name	The Name attribute of the object SHALL map to *name* attribute of the appropriate derived BPEL4WS element (as per mappings described in the sections below). The extra spaces and non-alphanumeric characters MUST be stripped from the Name to fit with the XML specification of the *name* attribute. Note that there may be two or more elements with the same name after the BPMN name has been stripped.
Assignments	Each Assignments Expression will map to a BPEL4WS *assign* activity. Refer to the section entitled "Assignment Mapping" on page 195 for more details about the mappings associated with the *assign* element.

A.1.4   Events

Start Event Mappings

The following table displays a set of mappings from the variations of a Start Event to BPEL4WS elements (these mappings extend the mappings common to objects--see Section A.1.3, "Common Flow Object Mappings," on page 147):

Table A.4 - Start Event Mappings to BPEL4W

Start Event	Mapping to BPEL4WS
EventType = Start and Trigger	The mapping to BPEL4WS is specific to the Trigger setting. These mappings are defined in the rows below.

Table A.4 - Start Event Mappings to BPEL4W

Start Event	Mapping to BPEL4WS
None	There is no BPEL4WS element that a Start Event will map to with a Trigger that is None. The object(s) that are the Target(s) of Sequence Flow that originate from the Start Event will determine the first BPEL4WS element of the Process. Note that a valid BPEL4WS *process* must begin with a *receive* or a *pick* activity that has a *createInstance* set to "yes." The *receive* or *pick* will likely be placed within a *sequence* or a *flow*.
Message	This will map to the *receive* element. The *createInstance* attribute of the *receive* element will be set to "yes."
Message	The Message attribute maps to the *variable* attribute of the *receive* activity. See "Messages" on page 195 for more information about how a BPMN Message maps to BPEL4WS and WSDL.
Implementation = Web Service	The Implementation attribute MUST be a Web service or MUST be converted to a Web Service for mapping to BPEL4WS. The Web Service Attributes are mapped as follows: The Participant attribute is mapped to the *partnerLink* attribute of the BPEL4WS activity. The Interface attribute is mapped to the *portType* attribute of the BPEL4WS activity. The Operation attribute is mapped to the *operation* attribute of the BPEL4WS activity.
Timer	This will map to the *receive* element. The *createInstance* attribute of the *receive* element will be set to "yes." The remaining attributes of the *receive* will be mapped as shown for the Message Start Event (see above). The functionality of the timing as defined in the Start Event must be implemented in a separate process that will start itself, then use a *wait* element for the defined time, and then use an *invoke* to send a message that will be received by the above *receive* element. A specific Message and Web service implementation must be provided so that the mappings to *receive* element can be completed.
Conditional	This will map to the *receive* element. The *createInstance* attribute of the *receive* element will be set to "yes." The remaining attributes of the *receive* will be mapped as the same way as for the Message Start Event (see above). **Note**: the Message is expected to arrive from the application that tracks and triggers Business Rules.
Multiple	This will map to a BPEL4WS *pick* will be required to process the messages with a separate *onMessage* for each defined Trigger. The *createInstance* attribute of the *pick* element will be set to "yes." This means that a single instance of the process will be instantiated when the first message received through the *pick onMessage* is triggered. The onMessage mappings are the same as that of a receive and as defined for the Message Start Event (see above).
With Assignments Expression	Each Assignments Expression will map to a BPEL4WS *assign* that will follow the *receive*. See Section A.1.15, "Assignment Mapping," on page 195 for more details about the mappings associated with the *assign* element.

End Event Mappings

The following table displays a set of mappings from the variations of a End Event to BPEL4WS elements (these mappings extend the mappings common to objects--see Section A.1.3, "Common Flow Object Mappings," on page 147):

Table A.5 - End Event Mappings to BPEL4WS

End Event	Mapping to BPEL4WS
EventType = End and Result	The mapping to BPEL4WS is specific to the Result setting. These mappings are defined in the rows below.
None	There is no BPEL4WS element that a End Event will map to with a Result that is None. However, it marks the end of a path within the Process and will be used to define the boundaries of complex BPEL4WS elements. The object(s) that are the Source(s) of Sequence Flow that Target the End Event will determine the final BPEL4WS elements of the Process.
Message	This will map to a BPEL4WS *reply* or an *invoke*. The appropriate BPEL4WS activity will be determined by the implementation defined for the Event. That is, the *portType* and *operation* of the Message will be used to check to see if an upstream Message Event have the same *portType* and *operation*. If these two attributes are matched, then the Event will map to a *reply*, if not, the Event will map to an *invoke*.
Message	The Message attribute maps to the *variable* attribute of the *reply* or the *outputVariable* of the *invoke*. See "Messages" on page 195 for more information about how a BPMN Message maps to BPEL4WS and WSDL.
Implementation = Web Service	The Implementation attribute MUST be a Web service or MUST be converted to a Web Service for mapping to BPEL4WS. The Web Service Attributes are mapped as follows: The Participant attribute is mapped to the *partnerLink* attribute of the BPEL4WS activity. The Interface attribute is mapped to the *portType* attribute of the BPEL4WS activity. The Operation attribute is mapped to the *operation* attribute of the BPEL4WS activity.
Error	This will map to a *throw* element. The ErrorCode attribute of the Event will map to the *faultName* attribute of the *throw*.
Cancel	The mapping of the Cancel Intermediate Event to BPEL4WS is an open issue.
Compensation	This will map to a *compensate* element. The Name of the activity referenced by the Compensation Event will map to the *scope* attribute of the *compensate* element.
Terminate	This will map to the *terminate* element.
Multiple	This will map to a this will map to a combination of *invoke*, *throw*, *fault*, and *compensation* elements as they are defined above.
With Assignments Expression	This will map to a BPEL4WS *assign* that will precede any other mappings required by the Event. See Section A.1.15, "Assignment Mapping," on page 195 for more details about the mappings associated with the *assign* element.

Intermediate Event Mappings

The following table displays a set of mappings from the variations of a Intermediate Event to BPEL4WS elements (these mappings extend the mappings common to objects--see Section A.1.3, "Common Flow Object Mappings," on page 147):

Table A.6 - Intermediate Event Mappings to BPEL4WS

Intermediate Event	Mapping to BPEL4WS
EventType = Intermediate and Trigger	The mapping to BPEL4WS is specific to the Trigger setting. These mappings are defined in the sections below.
With Assignments Expression	this will map to a BPEL4WS *assign*. See Section A.1.15, "Assignment Mapping," on page 195 for more details about the mappings associated with the *assign* element.

None Intermediate Events

The mappings for None Intermediate Events are described in the following table (these mappings extend the mappings common to Intermediate Events--see Section , "Intermediate Event Mappings," on page 150):

Table A.7 - None Intermediate Mappings to BPEL4WS

Intermediate Event	Mapping to BPEL4WS
Trigger = None	There is no BPEL4WS element that a Intermediate Event will map to with a Trigger that is None. These types of Intermediate Events are often used for documentation purposes to show a specific state of the Process.

Message Intermediate Events

The mappings for Message Intermediate Events are described in the following table (these mappings extend the mappings common to Intermediate Events--refer to the section entitled "Intermediate Event Mappings" on page 150):

Table A.8 - Message Intermediate Mappings to BPEL4WS

Intermediate Event	Mapping to BPEL4WS
Trigger = Message	This mapping is defined in the next five (5) rows.
Within the Normal Flow	If the Participant defined in the To attribute of the Message is the same Participant as that of the Process that contains the Event, then this will map to a *receive*. The *createInstance* attribute of the *receive* element will be set to "no." If the Participant defined in the From attribute of the Message is the same Participant as that of the Process that contains the Event, then this will map to a (one-way) *invoke*.
Message	The Message attribute maps to the *variable* attribute of the *reply* or the *outputVariable* of the *invoke*. See "Messages" on page 195 for more information about how a BPMN Message maps to BPEL4WS and WSDL.
Implementation = Web Service	The Implementation attribute MUST be a Web service or MUST be converted to a Web Service for mapping to BPEL4WS. The Web Service Attributes are mapped as follows: The Participant attribute is mapped to the *partnerLink* attribute of the BPEL4WS activity. The Interface attribute is mapped to the *portType* attribute of the BPEL4WS activity. The Operation attribute is mapped to the *operation* attribute of the BPEL4WS activity.

Table A.8 - Message Intermediate Mappings to BPEL4WS

Intermediate Event	Mapping to BPEL4WS
Without an incoming Sequence Flow (but not attached to an Activity Boundary)	The Participant defined in the To attribute of the Message MUST be the same Participant as that of the Process that contains the Event. The *process* will be given a *scope* (if it doesn't already have one). A *eventHandler*s element will be defined for the *scope*. An *onMessage* element will be added to the *eventHandler*s element.
Message	The Message attribute maps to the *variable* attribute of the *onMessage*. See "Messages" on page 195 for more information about how a BPMN Message maps to BPEL4WS and WSDL.
Implementation = Web Service	The Implementation attribute MUST be a Web service or MUST be converted to a Web Service for mapping to BPEL4WS. The Web Service Attributes are mapped as follows: The Participant attribute is mapped to the *partnerLink* attribute of the *onMessage*. The Interface attribute is mapped to the *portType* attribute of the *onMessage*. The Operation attribute is mapped to the *operation* attribute of the *onMessage*.
Attached to an Activity Boundary	The mappings of the activity (to which the Event is attached) will be placed within a *scope*. A *faultHandler*s element will be defined for the *scope*. A *catch* element will be added to the *faultHandlers* element with "<message name>_Exit" as the *faultName* attribute. An *eventHandler*s element will be defined for the *scope*. The Event will map to an *onMessage* element within the *eventHandler*s. The mapping to the *onMessage* attributes is the same as described for the *receive* above. The activity for the *onMessage* will be a *throw* with "<message name>_Exit" as the *faultName* attribute.
Used in an Event-Based Decision	This will map to an *onMessage* within a *pick*.The mapping to the *onMessage* attributes is the same as described for the *receive* above.

Timer Intermediate Events

The mappings for Timer Intermediate Events are described in the following table (these mappings extend the mappings common to Intermediate Events--see Section , "Intermediate Event Mappings," on page 150):

Table A.9 - Timer Intermediate Mappings to BPEL4WS

Intermediate Event	Mapping to BPEL4WS
Trigger = Timer	This mapping is defined in the next three (3) rows.
Within the Normal Flow	This will map to a *wait*. The TimeDate attribute maps to the *until* attribute of the *wait*. The TimeCycle attribute maps to the *for* attribute of the *wait*.
Without an incoming Sequence Flow (but not attached to an Activity Boundary)	The *process* will be given a *scope* (if it doesn't already have one). A *eventHandler*s element will be defined for the *scope*. An *onAlarm* element will be added to the *eventHandler*s element. The TimeDate attribute maps to the *until* attribute of the *onAlarm*. The TimeCycle attribute maps to the *for* attribute of the *onAlarm*.

Table A.9 - Timer Intermediate Mappings to BPEL4WS

Intermediate Event	Mapping to BPEL4WS
Attached to an Activity Boundary	The mappings of the activity (to which the Event is attached) will be placed within a *scope*. A *faultHandler*s element will be defined for the *scope*. A *catch* element will be added to the *faultHandlers* element with "<Event name>_Exit" as the *faultName* attribute. An *eventHandler*s element will be defined for the *scope*. The Event will map to an *onAlarm* element within the *eventHandler*s. The TimeDate attribute maps to the *until* attribute of the *onAlarm*. The TimeCycle attribute maps to the *for* attribute of the *onAlarm*. The activity for the *onAlarm* will be a *throw* with "<message name>_Exit" as the *faultName* attribute.
Used in an Event-Based Decision	This will map to an *onAlarm* within a *pick*. The TimeDate attribute maps to the *until* attribute of the *onAlarm*. The TimeCycle attribute maps to the *for* attribute of the *onAlarm*.

Error Intermediate Events

The mappings for Error Intermediate Events are described in the following table (these mappings extend the mappings common to Intermediate Events--see Section , "Intermediate Event Mappings," on page 150):

Table A.10 - Error Intermediate Mappings to BPEL4WS

Intermediate Event	Mapping to BPEL4WS
Trigger = Error	This mapping is defined in the next two (2) rows.
Within the Normal Flow	This will map to a *throw* element.
Attached to an Activity Boundary	The mappings of the activity (to which the Event is attached) will be placed within a *scope*. This Event will map to a *catch* element within a *scope*. If the Error Event does not have an ErrorCode, then a *catchAll* element will be added to the *faultHandlers* element. If the Error Event does has an ErrorCode, then a *catch* element will be added to the *faultHandlers* element with the ErrorCode mapping to the *faultName* attribute.

Cancel Intermediate Events

The mappings for Cancel Intermediate Events are described in the following table (these mappings extend the mappings common to Intermediate Events--see Section , "Intermediate Event Mappings," on page 150):

Table A.11 - Cancel Intermediate Mappings to BPEL4WS

Intermediate Event	Mapping to BPEL4WS
Trigger = Cancel	The mapping of the Cancel Intermediate Event to BPEL4WS is an open issue.

Conditional Intermediate Events

The mappings for Conditional Intermediate Events are described in the following table (these mappings extend the mappings common to Intermediate Events--see Section , "Intermediate Event Mappings," on page 150):

Table A.12 - Conditional Intermediate Mappings to BPEL4WS

Intermediate Event	Mapping to BPEL4WS
Trigger = Conditional	This mapping is defined in the next two (2) rows.
Within the Normal Flow	This will map to the *receive* element. The *createInstance* attribute of the *receive* element will be set to "no." The remaining attributes of the *receive* will be mapped as shown for the Message Start Event (see above).
Without an incoming Sequence Flow (but not attached to an Activity Boundary)	The Participant defined in the To attribute of the Message MUST be the same Participant as that of the Process that contains the Event. The *process* will be given a *scope* (if it doesn't already have one). A *eventHandlers* element will be defined for the *scope*. The Event will map to an *onMessage* element within the *eventHandlers*. The mapping to the *onMessage* attributes is the same as described for the *receive* for the Message Event above. **Note**: the Message is expected to arrive from the application that tracks and triggers Business Rules.
Attached to an Activity Boundary	The mappings of the activity (to which the Event is attached) will be placed within a *scope*. A *faultHandlers* element will be defined for the *scope*. A *catch* element will be added to the *faultHandlers* element with "<message name>_Exit" as the *faultName* attribute. An *eventHandlers* element will be defined for the *scope*. The Event will map to an *onMessage* element within the *eventHandlers*. The mapping to the *onMessage* attributes is the same as described for the *receive* for the Message Event above. **Note**: the Message is expected to arrive from the application that tracks and triggers Business Rules. The activity for the *onMessage* will be a *throw* with "<message name>_Exit" as the *faultName* attribute.
Used in an Event-Based Decision	This will map to an *onMessage* element within a *pick*. The mapping to the *onMessage* attributes is the same as described for the *receive* for the Message Event above.

Compensation Intermediate Events

The mappings for Compensation Intermediate Events are described in the following table (these mappings extend the mappings common to Intermediate Events--see Section , "Intermediate Event Mappings," on page 150):

Table A.13 - Compensation Intermediate Mappings to BPEL4WS

Intermediate Event	Mapping to BPEL4WS
Trigger = Compensation	This mapping is defined in the next two (2) rows.

Table A.13 - Compensation Intermediate Mappings to BPEL4WS

Intermediate Event	Mapping to BPEL4WS
Within the Normal Flow	This will map to a *compensate* element.The Name of the activity referenced by the Compensation Event will map to the *scope* attribute of the *compensate* element.
Attached to an Activity Boundary	The activity (to which the Event is attached) will be placed within a *scope*. This Event map to an *compensationHandler* element within a *scope*.

Link Intermediate Events

Link Intermediate Events are treated as "virtual Sequence Flow" that help connect the object preceding the source Link Event to the object following the target Link Event. Thus, the Link Intermediate Events are transparent to the BPEL4WS mapping (see the Section , "Handling Link Events as Go To Objects," on page 206):

Multiple Intermediate Events

The mappings for Multiple Intermediate Events are described in the following table (these mappings extend the mappings common to Intermediate Events--see Section , "Intermediate Event Mappings," on page 150):

Table A.14 - Multiple Intermediate Mappings to BPEL4WS

Intermediate Event	Mapping to BPEL4WS
Trigger = Multiple	This will map to a this will map to a combination of the mappings as they are defined in the Intermediate Event sections above.

A.1.5   Activities

Common Activity Mappings

The following table displays a set of mappings from the variations of activities to BPEL4WS elements (these mappings extend the mappings common to objects -- see Section A.1.3, "Common Flow Object Mappings," on page 147 -- Note that Table A.16 contains additional mappings that must be included within this set if extended by any other mapping table):

Table A.15 - Common Activity Mappings to BPEL4WS

Activity	Mapping to BPEL4WS

Properties	The set of Properties of an activity, as a whole, will map to a BPEL4WS *variable*. The *variable* element will be structured as follows:
	
	`<variable name="[activity.Name]_ActivityData"`
	`      messageType="[activity.Name]_ActivityDataMessage" />`
	
	The individual Properties will map to the *parts* of a WSDL *message*. The *message* element will be structured as follows:
	
	`<message name="[activity.Name]_ActivityDataMessage" >`
	` <part name="[Property.Name]"`
	`      type="xsd:[Property.Type]" />`
	`</message>`
	
	There will be as many *parts* to the *message* as there are Properties in the input group.

With Assignments Expression	This will map to a BPEL4WS *assign*. Refer to the section entitled "Assignment Mapping" on page 195 for more details about the mappings associated with the *assign* element.
AssignTime = Start	A BPEL4WS sequence will be created and the *assign* will precede
AssignTime = End	A BPEL4WS sequence will be created and the *assign* will follow

Activity Loop Mapping

The mapping to BPEL4WS for looping activities is complex and is made up of a number of activities that will surround the original mapping of the activity itself (which may be complex). The description of this mapping is divided into three sections to describe the basic setup of the loop (common to all loops), then the details of Standard looping, then the details of Multi-Instance looping.

Basic Loop Setup

The basic set up mappings, which are common to both Standard and Multi-Instance looping activities, are described in the following table (these mappings extend the mappings common to objects--see Section , "Common Activity Mappings," on page 155):

Table A.16 - Basic Activity Loop Mappings to BPEL4WS

Looping	Mapping to BPEL4WS
Activities with internal looping	Activities that have either a Standard or MultiInstance loop setting will result in a pattern of BPEL4WS elements, depending on the exact settings. This pattern will be placed within a BPEL4WS *sequence* activity. The details of the other mappings are described in the rows that follow.

Table A.16 - Basic Activity Loop Mappings to BPEL4WS

Looping	Mapping to BPEL4WS

LoopCounter	This attribute will map to a BPEL4WS *variable*, which will be part of the *process* definition. The variable will be structured as follows:
	
	``xml
	<variable name="[activity.Name]_loopCounter"
	         messageType="loopCounterMessage" />
	``
	
	Note: The LoopCounter mappings described in the this and the next three rows are only required for Multi-Instance loops and Standards loops that use the LoopMaximum attribute. For all looping activities, the LoopCounter can be used for reporting purposes.

Supporting WSDL Message	A WSDL *message* element will have to be created to support this *variable*. This *message* can be used for multiple *variables*. The *message* will be structured as follows:
	
	``xml
	<message name="loopCounterMessage" >
	  <part name="loopCounter" type="xsd:integer" />
	</message>
	``

Initialization of the LoopCounter	An *assign* activity will be created to initialize the *variable* before the start of the loop. This activity precede the *while* activity. This will be the first activity within the *sequence* activity. The *assign* will be structured as follows:
	
	``xml
	<assign name="[activity.Name]_initialize_loopCounter">
	  <copy>
	    <from expression="0"/>
	    <to variable="[activity.Name]_loopCounter"
	       part="loopCounter" />
	  </copy>
	</assign>
	``

Incrementing the LoopCounter	An *assign* activity will be created to update the loopCounter *variable* at the end of the *while* activity (see below). This activity will the last activity of the *sequence* activity that is within the *while* activity. The *assign* will be structured as follows:
	
	``xml
	<assign name="[activity.Name]_increment_loopCounter">
	  <copy>
	    <from expression="
	        bpws:getVariableData([activity.Name]_loopCounter,
	        loopCount) + 1"/>
	    <to variable="[activity.Name]_loopCounter"
	       part="loopCounter" />
	  </copy>
	</assign>
	``

Standard Loops

The loop mappings for Standard loops are described in the following table (these mappings extend the mappings of the Basic Loop Setup--refer to the previous section):

Table A.17 - Standard Activity Loop Mappings to BPEL4WS

Looping	Mapping to BPEL4WS
LoopType = Standard	For a Standard Looping activity, the mapping of the base BPMN activity will be placed within a BPEL4WS *sequence* that is within a *while*, and this will follow the *assign* described in the Basic Loop Setup (see Figure A.1 and Example A.1). Section , "Sub-Process Mappings," on page 173 or the Section , "Task Mappings," on page 175 for details about how the base activity will be mapped to BPEL4WS.
LoopCondition	The LoopCondition, which MUST be a boolean expression, will be used as the *condition* attribute of the *while* element.The *while condition* be structured as follows: `<while condition="[loopCondition]">`
TestTime = After	An After TestTime will map to the BPEL4WS *while* activity. However, to insure that the Task is performed at least once (i.e., the functionality of an until loop), a copy of the mapping for BPMN activity will be performed first in a *sequence*, followed by the *while* (which will contain the original copy of the mapping for the BPMN activity).
TestTime = Before	A Before TestTime does not require any additional mappings.
LoopMaximum	Any value in Maximum will be appended to the LoopCondition. For example with a LoopCondition of "x < 0" and Maximum of 5 (loops), the final expression would be "(x < 0) and ([ActivityName].LoopCounter <= 5)."

This uses the MI_Condition attribute of the Multi-Instance activity

Continue the Process

This represents the BPEL4WS *sequence* used for the mapping of the multi-instance activity

<<assign>> [activity.Name]_ initialize_ loopCounter

[loopCondition]

Performed Mapped Activity

<<assign>> [activity.Name]_ increment_ loopCounter

This represents the BPEL4WS *while* generated from the multi-instance activity

There will be as many iterations of this activity as required until the loopCondition is satisfied

This may be a sequence of activities, depending on the mapping

This represents the BPEL4WS *sequence* within the *while* activity

Figure A.1 - BPMN Depiction of BPEL4WS Pattern for a Standard loop, TestTime = Before

Example A.1 displays sample BPEL4WS code that reflects the mapping of a Standard loop.

```xml
<!-- The Process data is defined first-->
<variable name="[activity.Name]_loopCounter" messageType="loopCounterMessage" />
<!-- The contents of the process prior to the looping activity are here-->
<sequence>
  <assign name="[activity.Name]_initialize_loopCounter">
    <copy>
      <from expression="0"/>
      <to variable="[activity.Name]_loopCounter" part="loopCounter" />
    </copy>
  </assign>
  <!-- If the TestTime is set to After, the mappings of the original activity
       are placed here, as well as within the while.-->
  <while condition="[loopCondition]">
    <sequence>

      <!--The mappings of the original activity are placed here.-->

      <assign name="[activity.Name]_increment_counter">
        <copy>
          <from expression="bpws:getVariableData([activity.Name]_loopCounter,loopCount)+1"/>
          <to variable="[activity.Name]_loopCounter" part="loopCounter" />
        </copy>
      </assign>
    </sequence>
  </while>
</sequence>
<!-- The contents of the process after the looping activity are here-->
```

Example A.1 - BPEL4WS Sample for a Standard Loop

Multi-Instance Loop Setup

The loop mappings for Multi-Instance loops are described in the following table (these mappings extend the mappings of the Basic Loop Settings--see "Basic Loop Setup" on page 155):

Table A.18 - Multi-Instance Activity Loop Setup Mappings to BPEL4WS

Multi-Instance	Mapping to BPEL4WS
LoopType = MultiInstance	For a Multi-Instance Looping activity, the mapping of the BPMN activity will be placed within a BPEL4WS *sequence* that is within a *while*, and this will follow the *assign* described in the Basic Loop Setup (see Figure A.1 and Example A.1). See Section , "Sub-Process Mappings," on page 173 or Section , "Task Mappings," on page 175 for details about how the base activity will be mapped to BPEL4WS.

Table A.18 - Multi-Instance Activity Loop Setup Mappings to BPEL4WS

Multi-Instance	Mapping to BPEL4WS

MI_Condition	This applies to both Sequential and Parallel MI_Ordering (see below).
	The MI_Condition, which MUST be a numeric expression, will map to an *assign* activity. This will be the first activity of the generated *sequence* activity (as described in the row above).
	First, a BPEL4WS *variable* must be created with a derived name and will have a structure as follows:
	
	`<variable name="[activity.Name]_forEachCount"`
	`        messageType="forEachCounterMessage" />`
	
	Second, an *assign* activity will be used to generate the number of instances that will be required. The *assign* will be structured as follows:
	
	`<assign name="[activity.Name]_determine_instances">`
	`  <copy>`
	`    <from expression="[MI_Condition Expression]"/>`
	`    <to variable="[activity.Name]_forEachCount"`
	`        part="forEachCount" />`
	`  </copy>`
	`</assign>`

Supporting WSDL Message	A WSDL *message* element will have to be created to support the *variable*. This *message* can be used for multiple *variables*. The *message* will be structured as follows:
	
	`<message name="forEachCounterMessage" >`
	`  <part name="forEachCount" part="xsd:integer" />`
	`</message>`

The condition for the *while*	The *condition* attribute of the *while* will be a derived expression that utilizes the loopCounter variable and compares it to the derived forEachCount (described in the row above). The *while condition* be structured as follows:
	
	`<while condition="`
	`      bpws:getVariableData([activity.Name]_loopCounter,`
	`       loopCounter) >=`
	`      bpws:getVariableData([activity.Name]_forEachCount,`
	`       forEachCount)">`

Sequential Multi-Instance Loops

The loop mappings for Sequential Multi-Instance loops are described in the following table (these mappings extend the mappings of the Multi-Instance Setup--refer to the section above):

Table A.19 - Sequential Multi-Instance Activity Loop Mappings to BPEL4WS

Multi-Instance	Mapping to BPEL4WS
MI_Ordering = Sequential	This type of looping utilizes both the Basic Loop Setup mappings and the above Multi-Instance mappings. No further mappings are necessary. See Figure A.2 and Figure A.2 for the complete mappings.

This represents the BPEL4WS **sequence** used for the mapping of the multi-instance activity

Continue the Process

<<assign>> [activity.Name]_ initialize_ loopCounter

<<assign>> [activity.Name]_ determine_ instances

loopCounter) >= forEachCount)

Performed Mapped Activity

<<assign>> [activity.Name]_ increment_ loopCounter

This uses the MI_Condition attribute of the Multi-Instance activity

This represents the BPEL4WS **while** generated from the multi-instance activity

There will be as many copies of this activity as there are instances as determine by the previous assign activity

This represents the BPEL4WS **sequence** within the *while* activity

Figure A.2 - BPMN Depiction of BPEL4WS Pattern for a Sequential Multi-Instance loop

Example A.2 displays some sample BPEL4WS code that reflects the mapping of a Standard loop.

```xml
<!-- The Process data is defined first-->
<variable name="[activity.Name]_loopCounter" messageType="loopCounterMessage" />
<variable name="[activity.Name]_forEachCount" messageType="forEachCounterMessage" />
<!-- The contents of the process prior to the looping activity are here-->
<sequence>
  <assign name="[activity.Name]_initialize_loopCounter">
    <copy>
      <from expression="0"/>
      <to variable="[activity.Name]_loopCounter" part="loopCounter" />
    </copy>
  </assign>
  <assign name="[activity.Name]_determine_instances">
    <copy>
      <from expression="[MI_Condition Expression]"/>
      <to variable="[activity.Name]_forEachCount" part="forEachCount" />
    </copy>
  </assign>
  <while condition="bpws:getVariableData([activity.Name]_loopCounter,loopCounter) >=
                    bpws:getVariableData([activity.Name]_forEachCount,forEachCount)">
    <sequence>

      <!--The mappings of the original activity are placed here.-->

      <assign name="[activity.Name]_increment_counter">
        <copy>
          <from expression="bpws:getVariableData([activity.Name]_loopCounter,loopCount)+1"/>
          <to variable="[activity.Name]_loopCounter" part="loopCounter" />
        </copy>
      </assign>
    </sequence>
  </while>
</sequence>
<!-- The contents of the process after the looping activity are here-->
```

Example A.2 - BPEL4WS Sample for a Multi-Instance Loop with Sequential Ordering

Parallel Multi-Instance Loop Setup

The loop mappings for Sequential Multi-Instance loops are described in the following table (these mappings extend the mappings of the Multi-Instance Setup--refer to the section above):

Table A.20 - Parallel Multi-Instance Activity Loop Mappings to BPEL4WS

Multi-Instance	Mapping to BPEL4WS
MI_Ordering = Parallel	A BPEL4WS *while* activity will also be used for Parallel ordering. However, since the Task is to be performed in parallel, the mapping to the Tasks cannot be contained within the *while*. To get the parallel behavior, each copy of the multi-instance Task will be placed into a separate, derived BPEL4WS *process*[1]. A one-way *invoke* will be used to "spawn" each *process* and, thus, each instance of the Task. Since the *invoke* is only one-way, and doesn't wait for a response from the *process*, the *invoke* will complete quickly and the *while* will cycle through all of its iterations quick enough that the instantiations of the Task mappings will be effectively, if not literally, in parallel. The setting for the MI_FlowCondition attribute will determine what BPEL4WS elements will follow the *while* activity. These mappings will be described in the next four sections.
The *while* condition	The *while* condition will be the same as that of the Sequential ordering (see previous section).
Spawning the process	In the *while* activity, a one-way *invoke* activity will be created and used to "spawn" each of the derived *processes*. The *name* attribute for each derived *invoke* will be in the following format: `<invoke name="Spawn_Process_For_[activity.Name]" ... >` This invoke will replace the mappings of the original activity, which was in the while for Standard loops and Sequential Multi-Instance Loops.
The spawned process	The derived *process* will start with a *receive* that accepts the message that is sent by the one-way *invoke* that is within the *while* loop of the original *process*. The name of the process will be `"Spawned_Process_For_[activity.Name]."` The original Task will be mapped and those BPEL4WS elements will follow the initial *receive*. After all the mapped elements have been completed, then a one-way *invoke* will be used to send a message back to the original *process* has a notification that the spawned *process* is completed. This will be the last element of the spawned *process* (see Figure A.3 and Example A.3). The *name* attribute for the derived *invoke* will be in the following format: `<invoke name="[activity.Name]_Completed" ... >`

Table A.20 - Parallel Multi-Instance Activity Loop Mappings to BPEL4WS

Multi-Instance	Mapping to BPEL4WS
Copying variables to/ from the spawned processes	Since the Parallel Multi-Instance Task mappings are going to be performed within the a different process instance, the variables of the original *process* will need to be passed to the spawned *process* through the *inputVariable* of the one-way *invoke* that spawns the *process*. Likewise, any variables that are updated in the spawned *process* will need to be passed back to the original *process* through the *inputVariable* of the one-way *invoke* that indicates that the spawned *process* has completed. Note: Once the individual derived processes are instantiated, they will be blind to any changes in process variables. From the BPMN point of view, all the multi-instance activities are within the same context as the original Process and, thus, should be able to utilize any dynamic changes to Process Properties (BPEL4WS variables) as they occur (this is especially true for multi-instance Sub-Processes). It is up to the BPEL4WS execution environment to provide a "virtual context" for all the derived processes to "share" the process variables.
Receiving completion messages	As mentioned above, the spawned *processes* will send a message back to the original *process* after it has completed performing the behavior of the original activity. A BPEL4WS *receive* activity will be used to receive the messages back from all the spawned *processes*. The settings of the MI_FlowCondition will determine The *name* attribute for each derived *receive* will be structured as follows: `<receive name="[activity.Name]_Completed" ... >` The setting of the MI_FlowCondition attribute will determine how many *receive* activities will be required. Once the appropriate number of messages have been received back from the spawned *processes*, the original *process* will continue.

1. Note: BPEL4WS does not have a sub-process capability. It is likely that sub-processes, both Embedded and Reference, will be added to BPEL4WS in the future. When this capability has been added, the mapping for derived processes will be updated.
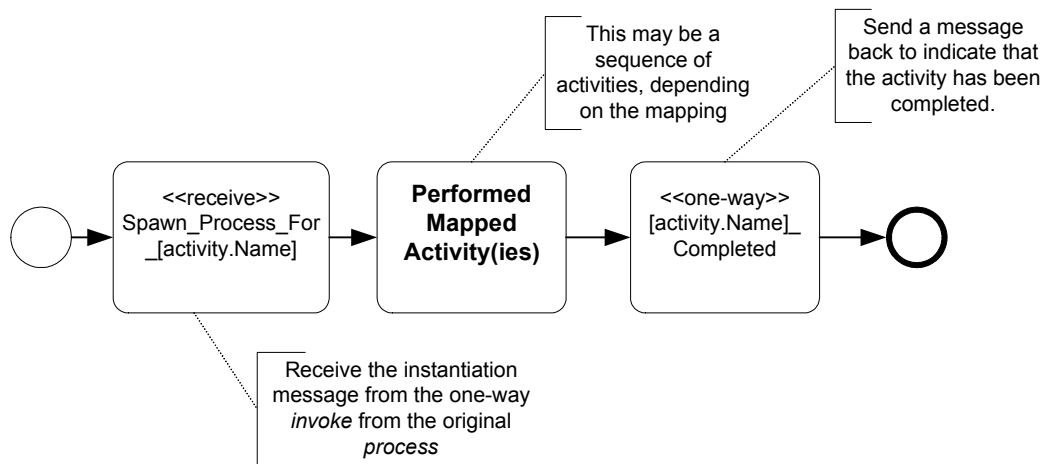
Figure A.3 - Structure of Process to be Spawned for Parallel Multi-instance

Example A.3 displays some sample BPEL4WS code that reflects the mapping of a Multi-Instance loop that has Parallel ordering and must synchronize all the looped activities.

```
<process name="Spawned_Process_For_[activity.Name]" ... >
  <sequence>
    <receive name="Spawn_Process_For_[activity.Name]" ... >

    <!--The mappings of the original activity are placed here.-->

    <invoke name="[activity.Name]_Completed" ... >
  </sequence>
</process>
```

Example A.3 - BPEL4WS Sample of a derived *process* for Parallel Multi-Instance loops

Parallel Multi-Instance Loops -- Flow Condition All

The loop mappings for Parallel Multi-Instance loops that have an MI_FlowCondition of All are described in the following table (these mappings extend the mappings of the Parallel Multi-Instance Setup--refer to the section above):

Table A.21 - Parallel Multi-Instance Activity, MI_FlowCondition = All

Multi-Instance	Mapping to BPEL4WS
MI_FlowCondition = All	This setting utilizes the mechanisms described above for the Parallel ordering. The "All" setting requires that all of the spawned *processes* must be completed before the original *process* can continue (see Figure A.4 and Example A.4).
Synchronizing the completion of the spawned processes	The synchronization from the spawned *processes* is managed through the messages sent by those *processes* when they have completed the behavior defined by the original activity. These messages will be received by the original *process* and when the messages from all the spawned *processes* are received, then the original *process* can continue. To ensure that all the messages are received, a second *while* activity will used. This while will contain a *receive* activity (for the completion messages) and an *assign* activity to increment the loop counter. The *while condition* attribute will be the same as the condition for the *while* that generated all the spawned *processes*, so that the same number of messages will be received as there were spawned *processes*.
Resetting the loop Counter	Prior to the second *while* activity, another *assign* will be required to reset the loop counter. The contents of the *assign* activity will be the same as the *assign* that originally initialized the loopCounter. The *name* attribute for the derived *assign* will be in the following format: `<assign name="[activity.Name]_reset_loopCounter" ... >`

This represents the
BPEL4WS
sequence
used for the mapping of the
multi-instance activity

**Continue the
Process**

This represents the
BPEL4WS
while
used to synchronize the
completion of the instances

This uses the
MI_Condition
attribute of the Multi-
Instance activity

<<assign>>
[activity.Name]_reset
_loopCounter

loopCounter) >=
forEachCount)

<<receive>>
[activity.Name]_
Completed

<<assign>>
[activity.Name]_
increment_
loopCounter

There will be as many
responses back from the
spawned process as
there are instances of
that process

This represents the
BPEL4WS
sequence
within the *while* activity

<<assign>>
[activity.Name]_
initialize_
loopCounter

<<assign>>
[activity.Name]_
determine_
instances

loopCounter) >=
forEachCount)

<<one-way>>
Spawn_Process_For
_[activity.Name]

<<assign>>
[activity.Name]_
increment_
loopCounter

This represents the
BPEL4WS
while
generated from the multi-
instance activity

There will be as many
copies of this activity as
there are instances as
determine by the
previous assign activity

This represents the
BPEL4WS
sequence
within the *while* activity

Figure A.4 - BPEL4WS Pattern of Parallel Multi-instance, MI_FlowCondition = All

```xml
<!-- The Process data is defined first-->
<variable name="[activity.Name]_loopCounter" messageType="loopCounterMessage" />
<variable name="[activity.Name]_forEachCount" messageType="forEachCounterMessage" />
<!-- The contents of the process prior to the looping activity are here-->
<sequence>
  <assign name="[activity.Name]_initialize_loopCounter">
    <copy>
      <from expression="0"/>
      <to variable="[activity.Name]_loopCounter" part="loopCounter" />
    </copy>
  </assign>
  <assign name="[activity.Name]_determine_instances">
    <copy>
      <from expression="[MI_Condition Expression]"/>
      <to variable="[activity.Name]_forEachCount" part="forEachCount" />
    </copy>
  </assign>
  <while condition=" bpws:getVariableData([activity.Name]_loopCounter,loopCounter) >=
                    bpws:getVariableData([activity.Name]_forEachCount,forEachCount)">
    <sequence>
      <invoke name=" Spawn_Process_For_[activity.Name]" ... >
      <assign name="[activity.Name]_increment_counter">
        <copy>
          <from expression="bpws:getVariableData([activity.Name]_loopCounter,loopCount)+1"/>
          <to variable="[activity.Name]_loopCounter" part="loopCounter" />
        </copy>
      </assign>
    </sequence>
  </while>
  <assign name="[activity.Name]_reset_loopCounter">
    <copy>
      <from expression="0"/>
      <to variable="[activity.Name]_loopCounter" part="loopCounter" />
    </copy>
  </assign>
  <!-- Set a while to receive all the return messages. The condition will be the same.-->
  <while condition=" bpws:getVariableData([activity.Name]_loopCounter,loopCounter) >=
                    bpws:getVariableData([activity.Name]_forEachCount,forEachCount)">
    <sequence>
      <receive name="[activity.Name]_Completed" ... >
      <assign name="[activity.Name]_increment_counter">
        <copy>
          <from expression="bpws:getVariableData([activity.Name]_loopCounter,loopCount)+1"/>
          <to variable="[activity.Name]_loopCounter" part="loopCounter" />
        </copy>
      </assign>
    </sequence>
  </while>
</sequence>
<!-- The contents of the process after the looping activity are here-->
```

Example A.4 - BPEL4WS Sample of a Parallel Multi-Instance Loop, MI_FlowCondition = All

Parallel Multi-Instance Loops -- Flow Condition One

The loop mappings for Parallel Multi-Instance loops that have a MI_FlowCondition of One are described in the following table (these mappings extend the mappings of the Parallel Multi-Instance Setup--refer to the section above):

Table A.22 - Parallel Multi-Instance Activity Loop, MI_FlowCondition = One

Multi-Instance	Mapping to BPEL4WS
MI_FlowCondition = One	This setting utilizes the mechanisms described above for the Parallel ordering. The "One" setting requires that only one of the spawned *processes* must be completed before the original *process* can continue (see Figure A.5 and Example A.5).
Receiving the completion message	Only one message is required from any one of the spawned *processes* before the original *process* can continue. Thus, there will be a single *receive* activity following the *while* activity. The *receive* will be the last element of the *sequence* that was started for the mapping of the Multi-Instance activity. The other spawned *processes* will continue there activities in parallel, but their completion will have no direct impact on the flow of the main process (their messages won't be received). Note: As mentioned above, it is up to the BPEL4WS execution environment to provide a "virtual context" for all the derived processes to "share" the process variables that may be updated by the spawned processes with the original process, even if there are no specific BPEL4WS activities to manage this information.
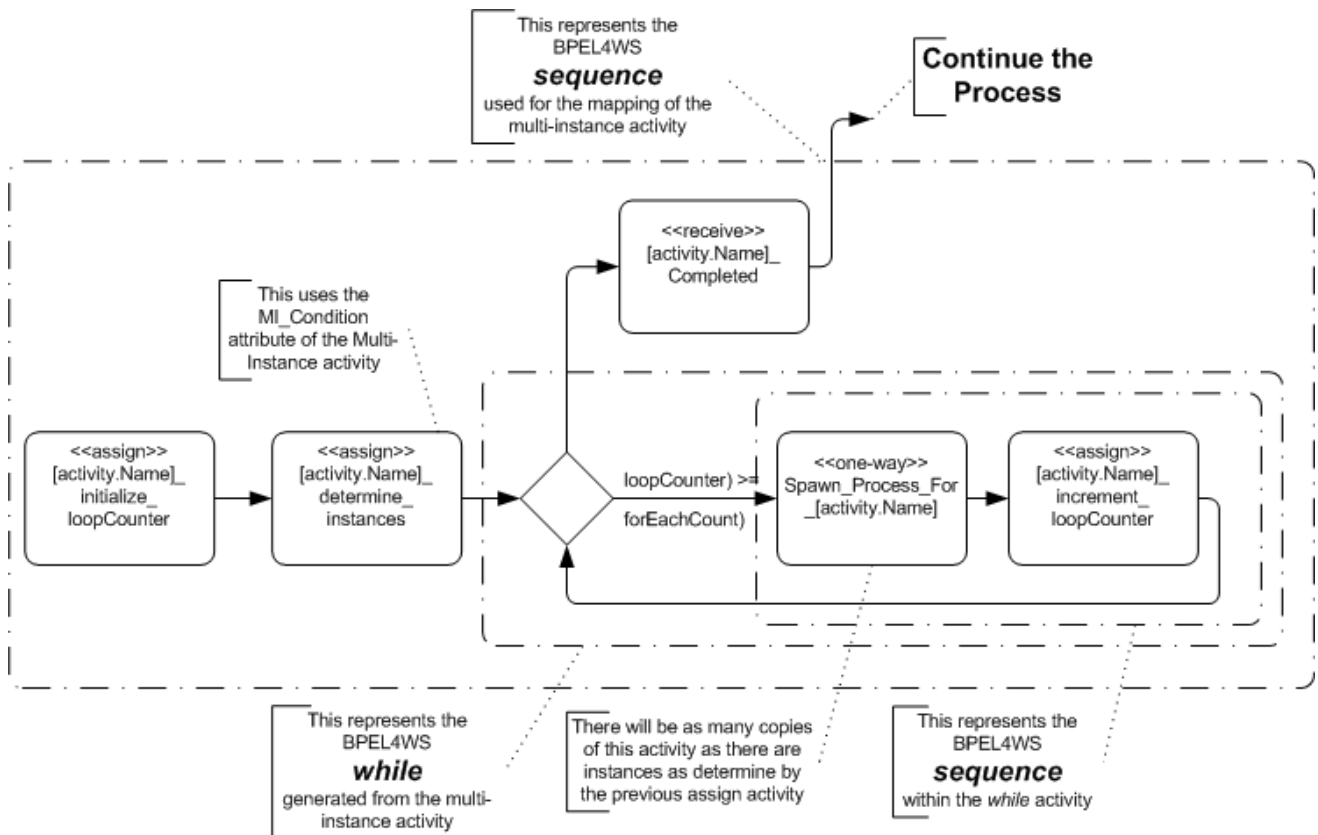
Figure A.5 - BPEL4WS Pattern of Parallel Multi-instance, MI_FlowCondition = One

Example A.5 displays some sample BPEL4WS code that reflects the mapping of a Multi-Instance loop that has Parallel ordering and must wait for only one of the looped activities.

```xml
<!-- The Process data is defined first-->
<variable name="[activity.Name]_loopCounter" messageType="loopCounterMessage" />
<variable name="[activity.Name]_forEachCount" messageType="forEachCounterMessage" />
<!-- The contents of the process prior to the looping activity are here-->
<sequence>
  <assign name="[activity.Name]_initialize_loopCounter">
    <copy>
      <from expression="0"/>
      <to variable="[activity.Name]_loopCounter" part="loopCounter" />
    </copy>
  </assign>
  <assign name="[activity.Name]_determine_instances">
    <copy>
      <from expression="[MI_Condition Expression]"/>
      <to variable="[activity.Name]_forEachCount" part="forEachCount" />
    </copy>
  </assign>
  <while condition="bpws:getVariableData([activity.Name]_loopCounter,loopCounter) >=
                    bpws:getVariableData([activity.Name]_forEachCount,forEachCount)">
    <sequence>

      <!--The mappings of the original activity are placed here.-->

      <assign name="[activity.Name]_increment_counter">
        <copy>
          <from expression="bpws:getVariableData([activity.Name]_loopCounter,loopCount)+1"/>
          <to variable="[activity.Name]_loopCounter" part="loopCounter" />
        </copy>
      </assign>
    </sequence>
  </while>
  <receive name="[activity.Name]_Completed" ... >
</sequence>
<!-- The contents of the process after the looping activity are here-->
```

Example A.5 - BPEL4WS Sample of a Parallel Multi-Instance Loop, MI_FlowCondition = One

Parallel Multi-Instance Loops -- Flow Condition Complex

The loop mappings for Parallel Multi-Instance loops that have a MI_FlowCondition of Complex are described in the following table (these mappings extend the mappings of the Parallel Multi-Instance Setup--refer to the section above):

Table A.23 - Parallel Multi-Instance Activity Loop, MI_FlowCondition = Complex

Multi-Instance	Mapping to BPEL4WS
MI_FlowCondition = Complex	The mapping for this setting is almost the same as the MI_FlowCondition of All mapping (as described above) and seen in Figure A.4 and Example A.4). The difference is that the number of return messages required before the process flow will continue must be determined and the messages have been received.

Table A.23 - Parallel Multi-Instance Activity Loop, MI_FlowCondition = Complex

Multi-Instance	Mapping to BPEL4WS
The while condition for receiving completion messages	The second while in the sequence will be used to receive the appropriate number of completion messages. The ComplexMI_FlowCondition, which MUST be a boolean expression, will determine this number. The *while condition* be structured as follows: <code><while condition="[ComplexMI_FlowCondition]"></code>

Parallel Multi-Instance Loops -- Flow Condition None

The loop mappings for Parallel Multi-Instance loops that have a MI_FlowCondition of None are described in the following table (these mappings extend the mappings of the Parallel Multi-Instance Setup--refer to the section above):

Table A.24 - Parallel Multi-Instance Activity Loop, MI_FlowCondition = None

Multi-Instance	Mapping to BPEL4WS
MI_FlowCondition = None	This means that there is not synchronization or control of the Tokens that are generated through the multi-instance activity. This means that each Token will continue on independently and each Token will create a separate instantiation of each activity they encounter. Basically, it means there is a separate copy of the whole process, for each copy of the Multi-Instance activity, after that point. Each copy of the remainder of the process will continue independently. To create this behavior, the remainder of the process will moved into a new, derived *process*.
Spawning the rest of the process	This *process* will be spawned through a one-way *invoke* that will be placed within the *while* activity, after the mappings of the original BPMN activity (see Figure Figure A.6 and Example A.6). The *name* attribute for the derived *invoke* will be in the following format: <code><invoke name=</code> <code> "Spawn_Remainder_of_Process_from_[activity.Name]"...></code>
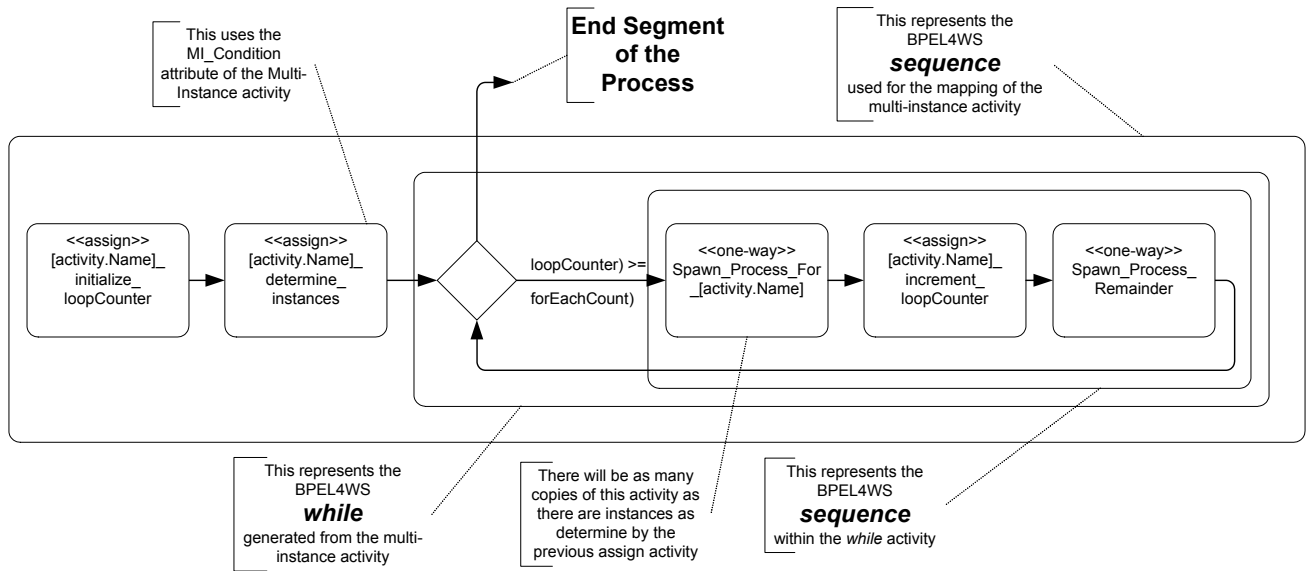
Figure A.6 - BPEL4WS Pattern of Parallel Multi-instance, MI_FlowCondition = None

Example A.6 displays some sample BPEL4WS code that reflects the mapping of a Multi-Instance loop that has Parallel ordering and must wait for none of the looped activities.

```
<!-- The Process data is defined first-->
<variable name="[activity.Name]_loopCounter" messageType="loopCounterMessage" />
<variable name="[activity.Name]_forEachCount" messageType="forEachCounterMessage" />
<!-- The contents of the process prior to the looping activity are here-->
<sequence>
  <assign name="[activity.Name]_initialize_loopCounter">
    <copy>
      <from expression="0"/>
      <to variable="[activity.Name]_loopCounter" part="loopCounter" />
    </copy>
  </assign>
  <assign name="[activity.Name]_determine_instances">
    <copy>
      <from expression="[MI_Condition Expression]"/>
      <to variable="[activity.Name]_forEachCount" part="forEachCount" />
    </copy>
  </assign>
  <while condition=" bpws:getVariableData([activity.Name]_loopCounter,loopCounter) >=
                    bpws:getVariableData([activity.Name]_forEachCount,forEachCount)">
    <sequence>

      <!--The mappings of the original activity are placed here.-->

      <assign name="[activity.Name]_increment_counter">
        <copy>
          <from expression="bpws:getVariableData([activity.Name]_loopCounter,loopCount)+1"/>
          <to variable="[activity.Name]_loopCounter" part="loopCounter" />
        </copy>
      </assign>
    </sequence>
  </while>
  <invoke name="Spawn_Remainder_of_Process_from_[activity.Name]" ... >
</sequence>
<!-- The contents of the process after the looping activity are here-->
```

Example A.6 - BPEL4WS Sample of a Parallel Multi-Instance Loop, MI_FlowCondition = None

Sub-Process Mappings

The following table displays a set of mappings from the variations of a Sub-Process to BPEL4WS elements (This extends the mappings that are defined for all activities--refer to the section entitled "Common Activity Mappings" on page 155):

Table A.25 - Sub-Process Mappings to BPEL4WS

Sub-Process	Mapping to BPEL4WS
ActivityType = SubProcess	The SubProcessType attribute will determine the exact mapping of a Sub-Process. See the next two sub-sections for these mappings.
IsATransaction	The mapping of a Sub-Process set to a Transaction is an Open Issue. Thus, there is no mapping defined when the IsATransaction is set to True, or the sub-attributes TransactionId, TransactionProtocol, and TransactionMethod.

Embedded Sub-Process

The following table displays a set of mappings from the variations of an Embedded Sub-Process to BPEL4WS elements (This extends the mappings that are defined for all activities--Section , "Sub-Process Mappings," on page 173):

Table A.26 - Embedded Sub-Process Mappings to BPEL4WS

Sub-Process	Mapping to BPEL4WS
SubProcessType = Embedded	This will map to a BPEL4WS *scope* element. The scope is not an independent *process* and will share the *process variables* of the higher-level *process*.
GraphicalElements	This is a list of all the graphical elements contained within the Process. Each of these elements will have their mapping, as defined in the sections below.
Adhoc	Ad Hoc Processes are not executable. Thus, this attribute MUST be set to False if the Process is to be mapped to BPEL4WS. The AdHocCompletionCondition and the AdHocOrdering attributes are only valid if the AdHoc attribute is True. Thus, these attributes will not be mapped to BPEL4WS.

Reusable Sub-Process

The following table displays a set of mappings from the variations of an Reusable Sub-Process to BPEL4WS elements (This extends the mappings that are defined for all activities--see Section , "Sub-Process Mappings," on page 173):

Table A.27 - Reusable Sub-Process Mappings to BPEL4WS

Task	Mapping to BPEL4WS
SubProcessType = Reusable	BPEL4WS does not have a sub-process element. Thus, Reusable Sub-Processes MUST map to a BPEL4WS *process*. That is, the contents of the Sub-Process, whether it is expanded or collapsed, will be contained within a separate *process*. The DiagramRef and ProcessRef attributes will identify the *process* that will be used for the mapping to the BPEL4WS *process*. The attributes of the other BPEL4WS *process* element will be filled from the mapping of the referenced Process. Section A.1.2, "Business Process Mappings," on page 146 for the details of this mapping. The Sub-Process object itself maps to an *invoke* activity that "calls" the *process*.
InputPropertyMaps	This attribute is actually a mapping of Process Properties to the Process Properties of the Process being referenced by the Sub-Process Object. The OutputPropertyMaps attribute maps to the *inputVariable* attribute of the *invoke* activity. See "Messages" on page 195 for more information about how a BPMN Message maps to BPEL4WS and WSDL.
OutputPropertyMaps	This attribute is actually a mapping of Process Properties to the Process Properties of the Process being referenced by the Sub-Process Object. The InputPropertyMaps attribute maps to the *outputVariable* attribute of the *invoke* activity. See "Messages" on page 195 for more information about how a BPMN Message maps to BPEL4WS and WSDL.

Reference Sub-Process

The following table displays a set of mappings from the variations of a Reference Sub-Process to BPEL4WS elements:

Table A.28 - Reference Sub-Process Mappings to BPEL4WS

Task	Mapping to BPEL4WS
SubProcessType = Reference	This type of Sub-Process is not directly mapped to BPEL4WS, since BPEL4WS does not support this type of referencing. However, the Sub-Process will be used as a placeholder for the Sub-Process that will be mapped (see next row).
TaskRef: Task	This attribute references another Sub-Process in the Process. It is the referenced Sub-Process that will be mapped and the mappings will be placed in the location of the Reference Sub-Process. That is, another copy of the entire mapping of the referenced Sub-Process will be created in this location (the mappings will also exist in the referenced Sub-Process' original location).

Task Mappings

The following table displays a set of mappings from the variations of a Task to BPEL4WS elements (This extends the mappings that are defined for all activities--see Section , "Common Activity Mappings," on page 155):

Table A.29 - Task Mappings to BPEL4WS

Task	Mapping to BPEL4WS
ActivityType = Task	The TaskType attribute will determine the exact mapping of a Task. See the next eight (8) sub-sections for these mappings.
Web service Mappings	The Implementation attribute MUST be a Web service or MUST be converted to a Web Service for mapping to BPEL4WS. The Web Service Attributes are mapped as follows: The Participant attribute is mapped to the *partnerLink* attribute of the BPEL4WS activity. The Interface attribute is mapped to the *portType* attribute of the BPEL4WS activity. The Operation attribute is mapped to the *operation* attribute of the BPEL4WS activity.

Service Task

The following table displays a set of mappings from the variations of a Service Task to BPEL4WS elements:

Table A.30 - ServiceTask Mappings to BPEL4WS

Task	Mapping to BPEL4WS
TaskType = Service	This type of Task maps to an *invoke* activity.
InMessage	The InMessage attribute maps to the *inputVariable* attribute of the *invoke* activity. See "Messages" on page 195 for more information about how a BPMN Message maps to BPEL4WS and WSDL.
OutMessage	The OutMessage attribute maps to the *outputVariable* attribute of the *invoke* activity. See "Messages" on page 195 for more information about how a BPMN Message maps to BPEL4WS and WSDL.

Table A.30 - ServiceTask Mappings to BPEL4WS

Task	Mapping to BPEL4WS
Implementation = Web Service	This will map as defined in Table A.29.

Receive Task

The following table displays a set of mappings from the variations of a Receive Task to BPEL4WS elements (This extends the mappings that are defined for all Tasks--see Section , "Task Mappings," on page 175):

Table A.31 - Receive Task Mappings to BPEL4WS

Task	Mapping to BPEL4WS
TaskType = Receive	This type of Task maps to a *receive* activity.
Message: Message	The Message attribute maps to the *variable* attribute of the *receive* activity. See "Messages" on page 195 for more information about how a BPMN Message maps to BPEL4WS and WSDL.
Instantiate False : Boolean	If the Instantiate attribute of the Task is set to False, then the *createInstance* attribute of the *receive* will not be included or it will be set to "no." If the Instantiate attribute of the Task is set to True, then the *createInstance* attribute of the *receive* will be set to "yes."
Implementation = Web Service	This will map as defined in Table A.29.

Send Task

The following table displays a set of mappings from the variations of a Send Task to BPEL4WS elements:

Table A.32 - Send Task Mappings to BPEL4WS

Task	Mapping to BPEL4WS
TaskType = Send	This type of Task maps to a *reply* or an *invoke* activity. The appropriate BPEL4WS activity will be determined by the implementation defined for the Task. That is, the *portType* and *operation* of the Task will be used to check to see if an upstream Receive Task have the same *portType* and *operation*. If these two attributes are matched, then the Send Task will map to a *reply*, if not, the Send Task will map to an *invoke*.
Message: Message	The Message attribute maps to the *variable* attribute of the *reply* activity or it maps to the *inputVariable* attribute of the *invoke* activity. See "Messages" on page 195 for more information about how a BPMN Message maps to BPEL4WS and WSDL.
Implementation = Web Service	This will map as defined in Table A.29.

User Task

The following table displays a set of mappings from the variations of a User Task to BPEL4WS elements:

Table A.33 - User Task Mappings to BPEL4WS

Task	Mapping to BPEL4WS
TaskType = User	This type of Task maps to an *invoke* activity.
Performers: String	The Performers is information needed by the implementation. Thus, it will be included in the InMessage being sent to the Web service, through the *inputVariable* attribute of the *invoke* activity.
InMessage	The InMessage attribute maps to the *inputVariable* attribute of the *invoke* activity. See "Messages" on page 195 for more information about how a BPMN Message maps to BPEL4WS and WSDL.
OutMessage	The OutMessage attribute maps to the *outputVariable* attribute of the *invoke* activity. See "Messages" on page 195 for more information about how a BPMN Message maps to BPEL4WS and WSDL.
Implementation = Web Service	This will map as defined in Table A.29.

Script Task

The following table displays a set of mappings from the variations of a Script Task to BPEL4WS elements:

Table A.34 - Script Task Mappings to BPEL4WS

Task	Mapping to BPEL4WS
TaskType = Script	This type of Task maps to an *invoke* activity. Since this activity is performed by a process engine, the default settings of the engine must be used to determine the settings of the *invoke* activity. That is, *partnerLink*, *portType*, *operation*, *inputVariable*, and maybe *outputVariable* are derived by these default settings. The script itself is performed when the appropriate Web service of the process engine in invoked.

Manual Task

The Manual Task does not map to BPEL4WS. Thus, this type of Task should not be used in a Process that is intended to generate BPEL4WS code.

Reference Task

The following table displays a set of mappings from the variations of a Reference Task to BPEL4WS elements:

Table A.35 - Reference Task Mappings to BPEL4WS

Task	Mapping to BPEL4WS
TaskType = Reference	This type of Task is not directly mapped to BPEL4WS, since BPEL4WS does not support this type of referencing. However, the Task will be used as a placeholder for the Task that will be mapped (see next row).

Table A.35 - Reference Task Mappings to BPEL4WS

Task	Mapping to BPEL4WS
TaskRef: Task	This attribute references another Task in the Process. It is the referenced Task that will be mapped and the mappings will be placed in the location of the Reference Task. That is, another copy of the entire mapping of the referenced Task will be created in this location (the mappings will also exist in the referenced Task's original location).

None Task

The following table displays a set of mappings from the variations of a None Task to BPEL4WS elements:

Table A.36 - None Task Mappings to BPEL4WS

Task	Mapping to BPEL4WS
TaskType = None	This type of Task maps to an *empty* activity.

A.1.6   Gateways

Common Gateway Mappings

The following table displays a set of mappings are common for Gateways to BPEL4WS elements (these mappings extend the mappings common to objects -- see Section A.1.3, "Common Flow Object Mappings," on page 147):

Table A.37 - Common Gateway Mappings to BPEL4WS

Data-Based Exclusive Gateways	Mapping to BPEL4WS
Gateway	A Gateway will map to a variety of BPEL4WS elements (e.g., *switch*, *pick*, *flow*) and patterns of elements.
Incoming Flow	A Gateway, as with activities, is a location where Sequence Flow can converge. The convergence of Sequence Flow potentially marks the end of a BPEL4WS structured element, if the correct number of flow converge. See Section A.1.17, "Determining the Extent of a BPEL4WS Structured Element," on page 195 for more details on converging of Sequence Flow and their mapping to BPEL4WS.
Outgoing Flow	The mapping will begin at the location of the Gateway. The BPMN elements that follow the Gateway, until all the outgoing paths have converged, will be contained within the extent of the mapping (e.g., they will be placed with in a *sequence* within a *switch case*). The end of the mapping will be determined by the convergence of the paths, through a variety of mechanisms (see Section A.1.17, "Determining the Extent of a BPEL4WS Structured Element," on page 195).
Assignments associated with Gates	This will map to a BPEL4WS *assign*. See Section A.1.15, "Assignment Mapping," on page 195 for more details about the mappings associated with the *assign* element.

Exclusive

Data-Based

The following table displays a set of mappings from the variations of a Data-Based Exclusive Gateway to BPEL4WS elements (these mappings extend the mappings common to objects -- see Section , "Common Gateway Mappings," on page 178):

Table A.38 - Data-Based Exclusive Gateway Mappings to BPEL4WS

Data-Based Exclusive Gateways	Mapping to BPEL4WS
Gateway (GatewayType = Exclusive; ExclusiveType = Data)	The Gateway will map to a BPEL4WS *switch*.
MarkerVisible	This does not have a mapping to BPEL4WS. Its purpose is to determine whether or not a graphical marker will be displayed.
Incoming Flow	
Outgoing Flow	
Gates	Each Gate will map to a *case* of the *switch*. The *cases* will be listed in the *switch* in the same order as they are listed for the Gateway.
Condition for the Sequence Flow associated with the Gate	This will map to the *condition* for a *switch* case.
BPMN Elements that follow the Gate.	If there is more than one element that follows the Gate, and this includes Assignments for the Gate, then these elements will be placed inside a *sequence* activity after these elements have been mapped.
DefaultGate	This will map to the *otherwise* element of the *switch*.
BPMN Elements that follow the DefaultGate.	If there is more than one element that follows the DefaultGate, and this includes Assignments for the Gate, then these elements will be placed inside a *sequence* activity after these elements have been mapped.

Event-Based

To relate the Event-Based Exclusive Gateway to BPEL4WS, the Gateway diamond marks the location of a BPEL4WS *pick* and the Intermediate Events that follow the Decision become the event handlers of the *pick* or *choice*. The activities that follow the Intermediate Events become the contents of the *activity sets* for the event handlers. The boundaries of the activity sets is actually determined by the configuration of the process; that is, the boundaries extend to where all the alternative paths are finally joined together (which could be the end of the Process).

The following table displays a set of mappings from the variations of a Event-Based Exclusive Gateway to BPEL4WS elements (these mappings extend the mappings common to objects -- see Section , "Common Gateway Mappings," on page 178):

Table A.39 - Data-Based Exclusive Gateway Mappings to BPEL4WS

Event-Based Exclusive Gateways	Mapping to BPEL4WS
Gateway (GatewayType = Exclusive; ExclusiveType = Event)	This Gateway will map to a BPEL4WS *pick*. The elements of the *pick* will be determined by the targets of the outgoing Sequence Flow. The specific mappings are described in the rows below.
Instantiate	If the Instantiate attribute of the Gateway is set to False, then the *createInstance* attribute of the *pick* MUST NOT be included OR it MUST be set to "no." If the Instantiate attribute of the Gateway is set to True, then the *createInstance* attribute of the *pick* MUST be set to "yes."
Gate with Receive Task as Target	The Receive Task will map to an *onMessage* element within the *pick*. The attributes of the Receive Task will map to the appropriate elements of the *onMessage*, such as *operation* and *portType*. See "Receive Task" on page 176 for the mapping of the Receive Task. Note that the name of the Task does not have a corresponding attribute within the *onMessage* element.
Gate with Message Intermediate Event as Target	A Message Intermediate Event will map to an *onMessage* element within the *pick*. The attributes of the Message Intermediate Event will map to the appropriate elements of the *onMessage*, such as *operation* and *portType*. See Section , "Intermediate Event Mappings," on page 150 for the mapping of the Message Intermediate Event.
Gate with Timer Intermediate Event as Target	A Timer Intermediate Event, which is the Target of a Sequence Flow associated with the Gate, will map to an onAlarm element within the pick. The Timedate attribute of the Event will map to the until element of the *onAlarm* element. The Timecycle attribute of the Event will map to the for element of the *onAlarm* element.
Gate with Link Intermediate Event as Target	A Link Intermediate Event, in this situation, will be considered as the same as receiving a message from a process. Thus, this will map to an *onMessage* element within the *pick*. The attributes of the Message Intermediate Event will map to the appropriate elements of the *onMessage*, such as *operation* and *portType*. See Section , "Intermediate Event Mappings," on page 150 for the mapping of the Message Intermediate Event.
Gate with Conditional Intermediate Event as Target	A Conditional Intermediate Event, in this situation, will be considered as the same as receiving a message from a system that tracks and generates Conditional events. Thus, this will map to an *onMessage* element within the *pick*. The attributes of the Message Intermediate Event will map to the appropriate elements of the *onMessage*, such as *operation* and *portType*. See Section , "Intermediate Event Mappings," on page 150 for the mapping of the Message Intermediate Event.
BPMN Elements that follow the first target of a Gate.	If there is more than one element that follows the first target of a Gate, and this includes Assignments for the Gate, then these elements will be placed inside a *sequence* activity after these elements have been mapped.

Inclusive

The following table displays a set of mappings from the variations of a Inclusive Gateway to BPEL4WS elements (these mappings extend the mappings common to objects -- See Section , "Common Gateway Mappings," on page 178):

Table A.40 - Inclusive Gateway Mappings to BPEL4WS

Inclusive Gateways	Mapping to BPEL4WS
Gateway (GatewayType = Inclusive)	The Gateway will map to a set of BPEL4WS *switches* within a BPEL4WS *flow*. An additional *switch* will be required if the DefaultGate is used (see below)
Gates	Each Gate will map to a *switch*. Each *switch* will binary in nature. That is, each switch will have exactly one *case* and one *otherwise*.
Condition for the Sequence Flow associated with the Gate	This will map to the *condition* for the *switch* case.
BPMN Elements that follow the Gate.	If there is more than one element that follows the Gate, and this includes Assignments for the Gate, then these elements will be placed inside a *sequence* activity after these elements have been mapped. If a DefaultGate is used, then an *assign* activity will follow all the other mappings (see below for details).
The *otherwise* element for the *switch*	The *otherwise* element for each *switch* will contain an *empty* activity. However, if the DefaultGate is used, then
DefaultGate	The DefaultGate will map to a *switch*. However, by using the DefaultGate, the mapping to BPEL4WS is more complicated (see Figure A.7 and Example A.7). This is the path that is taken if none of the other paths are taken. Thus, the decision about whether the Default Gate should be taken will occur after all the other Gate decisions have been determined. This means that the *switch* for the DefaultGate will follow the *flow* activity generated for all the Gates of the Gateway. Also, a *sequence* activity must encompass the *flow* and the *switch*.

Create the tracking variable	A *variable* must be used so that the switch for the DefaultGate will know whether or not the Default BPMN path should be taken. To do this, a BPEL4WS *variable* must be created with a derived name and will have a structure as follows:
	 `<variable name="[Gateway.Name]_noDefaultRequired"`
	 `        messageType="noDefaultRequired" />`

Supporting WSDL Message	A WSDL *message* element will have to be created to support this *variable*. This *message* can be used for multiple *variables*. The *message* will be structured as follows:
	 `<message name="noDefaultRequired" >`
	 `  <part name="noDefault" type="xsd:boolean" />`
	 `</message>`

Table A.40 - Inclusive Gateway Mappings to BPEL4WS

Inclusive Gateways	Mapping to BPEL4WS

Initialization of the tracking variable	An *assign* activity will be created to initialize the *variable* before the start of the loop. This *assign* precede the *flow* activity that contains all the *switches* derived from the Gates. This will be the first activity within the *sequence* activity. The *assign* will be structured as follows:
	
	``xml
	<assign name="[Gateway.Name]_initialize_noDefault">
	  <copy>
	    <from expression="false"/>
	    <to variable="[Gateway.Name]_noDefaultRequired"
	        part="noDefault" />
	  </copy>
	</assign>
	``

The *switch cases*	The condition for the *switch case* will used the noDefaultRequired *variable* and will structured as follows:
	
	``xml
	<switch>
	  <case condition="bpws:getVariableProperty(
	        [Gateway.Name]_noDefaultRequired,noDefault)=true">
	    <sequence>
	
	<!--The mappings of the original activity are placed here.-->
	<!--An assign activity (see below) is placed here.-->
	
	    </sequence>
	  </case>
	  <otherwise>
	    <empty/>
	  </otherwise>
	</switch>
	``

BPMN Elements that follow the DefaultGate	If there is more than one element that follows the DefaultGate, and this includes Assignments for the Gate, then these elements will be placed inside a *sequence* activity after these elements have been mapped. An *assign* activity will be placed in the sequence after all the other mappings (see next row).

Setting of the tracking variable	If any of the *switches* within the flow passes the condition of the switch case, then the noDefaultRequired must be set to True. This will ensure that the DefaultGate switch will bypass the mapped activities for the BPMN Default Gate.
	An *assign* activity will be created to set the *variable* to True. This will be the last activity within the *sequence* activity within the *switch*. The *assign* will be structured as follows:
	
	``xml
	<assign name="[Gateway.Name]_set_noDefault">
	  <copy>
	    <from expression="true"/>
	    <to variable="[Gateway.Name]_noDefaultRequired"
	        part="noDefault" />
	  </copy>
	</assign>
	``

This represents the BPEL4WS
flow
that encloses the *switches* for
the Gates

This represents the BPEL4WS
while
generated from a Gate

This represents the BPEL4WS
sequence
within the *switch case*

There may be more
switches included within
the flow, but there must
be at least one.

This represents the BPEL4WS
while
generated from the
DefaultGate

Performed
Mapped
Activity(ies)

<<assign>>
[activity.Name]_
set_noDefault

[Sequence Flow
Condition]

Otherwise

<<assign>>
[activity.Name]_
initialize_
noDefault

noDefault = True

Otherwise

Performed
Mapped
Activity(ies)

This represents the BPEL4WS
sequence
that encloses the whole
pattern

This represents the BPEL4WS
while
generated from a Gate

This represents the BPEL4WS
sequence
within the *switch case*

Figure A.7 - BPEL4WS Pattern of Inclusive Decision with two (2) Gates and a DefaultGate

Example A.7 displays some sample BPEL4WS code that reflects the mapping of a Multi-Instance loop that has Parallel ordering and must synchronize all the looped activities.

```
<!-- The Process data is defined first-->
<variable name="[activity.Name]_loopCounter" messageType="loopCounterMessage" />
<!-- The contents of the process prior to the looping activity are here-->
<sequence>
  <assign name="[Gateway.Name]_initialize_noDefault">
    <copy>
      <from expression="false"/>
      <to variable="[Gateway.Name]_noDefaultRequired" part="noDefault" />
    </copy>
  </assign>
  <flow>
    <!--There will be as many copies of the switch below as there are Gates.-->
    <switch>
      <case condition="[Sequence Flow Condition">
        <sequence>

      <!--The mappings of the activities are placed here.-->

          <assign name="[Gateway.Name]_initialize_noDefault">
            <copy>
              <from expression="true"/>
              <to variable="[Gateway.Name]_noDefaultRequired" part="noDefault" />
            </copy>
          </assign>
        </sequence>
      </case>
      <otherwise>
        <empty/>
      </otherwise>
    </switch>
  </flow>
  <switch>
    <case condition=
          "bpws:getVariableProperty([Gateway.Name]_noDefaultRequired,noDefault)=true">
    <sequence>

      <!--The mappings of the activities are placed here.-->

    </sequence>
    </case>
    <otherwise>
      <empty/>
    </otherwise>
  </switch>
</sequence>
```

Example A.7 - BPEL4WS Sample for the Pattern for an Inclusive Decision with a DefaultGate

Complex

The behavior and usage of Complex Gateways have not been well enough established for a mapping to BPEL4WS to be defined.

Parallel

The following table displays a set of mappings from the variations of a Parallel Gateway to BPEL4WS elements (these mappings extend the mappings common to objects --see Section , "Common Gateway Mappings," on page 178):

Table A.41 - Parallel Gateway Mappings to BPEL4WS

Parallel Gateways	Mapping to BPEL4WS
Gateway (GatewayType = Parallel)	The Gateway will map to a BPEL4WS *flow*.

A.1.7   Pool

Pools do not have any specific Mapping to Execution Languages. However, a Pool is associated with a mapping to a specific lower level language. For example, one Pool may encompass a BPEL4WS document while another Pool might encompass B2B Collaboration process.

A.1.8   Lane

Lanes do not have any specific Mapping to Execution Languages. They are designed to help organize and communicate how activities are grouped in a business process.

A.1.9   Artifacts

As a general rule, Artifacts do not map to BPEL4WS elements. They provide detailed information about how data will interact with the Flow Objects and Flow of Processes.

However, Text Annotations can map to the *documentation* element of BPM execution languages. If the Annotation is associated with a Flow Object and that object has a straight-forward mapping to a BPM execution language element, then the text of the Annotation will be placed in the *documentation* element of that object. If there is no straight-forward mapping to a BPM execution language element, then the text of the Annotation will be appended to the *documentation* element of the *process*.

For any new Artifact that is added to a BPD through a modeling tool, it will have to be determined whether or not that Artifact maps to any BPEL4WS element.

A.1.10 Sequence Flow

A Sequence Flow may not have a specific mapping to a BPEL4WS in most situations. However, when there is a section of the Diagram that contains parallel activities, then Sequence Flow may map to the *link* element. Details of this mapping are TBD. In general, the set of Sequence Flow within a Pool will determine how BPEL4WS elements are derived and the boundaries of those elements.

The following table displays a set of mappings from Sequence Flow to BPEL4WS elements:

Table A.42 - Sequence Flow Mappings to BPEL4WS

Sequence Flow	Mapping to BPEL4WS
Sequence Flow	This MAY map to a BPEL4WS *link* element. The location of the Sequence Flow within the Process will determine how or if it is mapped to a *link*. Even if the Sequence Flow is not mapped to a link, attributes, such as Condition, will be mapped to BPEL4WS elements, as described below.
Id, Categories, and Documentation	These Elements do not map to any BPEL4WS elements or attributes.
Name	If the Sequence is not being mapped to a *link*: This attribute does not map to any BPEL4WS elements or attributes. If the Sequence is being mapped to a *link*: The Name attribute of the Process SHALL map to *name* attribute of the *link*. The extra spaces and non-alphanumeric characters MUST be stripped from the Name to fit with the XML specification of the *name* attribute. Note that there may be two or more elements with the same name after the BPMN name has been stripped.
Source	If the Sequence is not being mapped to a *link*: This attribute does not map to any BPEL4WS elements or attributes. If the Sequence is being mapped to a *link*: This mapping is described in the next four (4) Rows.
Source Object is an Activity (for a *link*)	The mapping of the source activity will now include a *source* element. The Name of the Sequence Flow will map to *linkName* attribute of the *source* element.The extra spaces and non-alphanumeric characters MUST be stripped from the Name to fit with the XML specification of the *name* attribute. Note that there may be two or more elements with the same name after the BPMN name has been stripped. For an exception to the location of the *source* element, see the description of the mapping for a ConditionExpression when the Source object is an Activity below.
Source Object is a Gateway (for a *link*)	This mapping is described in the next two (2) Rows.
The Gateway maps to an activity (e.g., *switch*)	This mapping is the same as if the source object is an activity (see above).
The Gateway does not map to an activity	This Sequence Flow will be essentially combined with one of the Gateway's incoming Sequence Flow. (There will be a separate *link* for each of the incoming Sequence Flow). The Source of the second Sequence will be used at the Source for the original Sequence Flow. Then, this mapping is the same as if the Source object is an activity (see above).
Target	If the Sequence is not being mapped to a *link*: This attribute does not map to any BPEL4WS elements or attributes. If the Sequence is being mapped to a *link*: This mapping is described in the next four (4) Rows.

Table A.42 - Sequence Flow Mappings to BPEL4WS

Sequence Flow	Mapping to BPEL4WS
Target Object is an Activity	The mapping of the target activity will now include a *target* element. The Name of the Sequence Flow will map to *linkName* attribute of the *target* element.The extra spaces and non-alphanumeric characters MUST be stripped from the Name to fit with the XML specification of the *name* attribute. Note that there may be two or more elements with the same name after the BPMN name has been stripped.
Target Object is a Gateway	This mapping is described in the next two (2) Rows.
The Gateway maps to an activity (e.g., *switch*)	This mapping is the same as if the target object is an activity (see above).
The Gateway does not map to an activity	This Sequence Flow will be essentially combined with one of the Gateway's outgoing Sequence Flow. (There will be a separate *link* for each of the outgoing Sequence Flow). The Target of the second Sequence will be used at the Target for the original Sequence Flow. Then, this mapping is the same as if the target object is an activity (see above).
ConditionType = None	If the Sequence is not being mapped to a *link*: This attribute does not map to any BPEL4WS elements or attributes. If the Sequence is being mapped to a *link*: This means that there is no condition placed on the transition between elements (through the link). Thus, there is nothing to be mapped to BPEL4WS.
ConditionType = Expression	This mapping is described in the next two (2) Rows.
Source Object is a Gateway	The mapping of the Sequence Flow in this situation is described in Section , "Exclusive," on page 179, Section , "Inclusive," on page 181, and Section , "Complex," on page 184.
Source Object is an Activity	Since a Sequence Flow MUST NOT have a Condition if the Source is an activity, unless there are multiple outgoing Sequence Flow, a BPEL4WS *flow* will be required and the Sequence Flow will map to a *link* element. An *empty* activity will be placed in the flow and will contain all the *source* elements. The ConditionExpression will then map to the *transitionCondition* attribute of the *source* element that is contained in the appropriate BPEL4WS activity (see a description of locating the source above).
ConditionType = Default	The mapping of the Sequence Flow in this situation is described in Section , "Exclusive," on page 179, Section , "Inclusive," on page 181, and Section , "Complex," on page 184.
Quantity 1 : Integer	The mapping of the Quantity attribute, if its value is greater than one (1), BPEL4WS is an open issue.

When to Map a Sequence Flow to a BPEL4WS Link

In many situations, a Sequence Flow will not map to a BPEL4WS *link* element.

- To connect activities that are listed in a BPEL4WS structured activity (e.g., a *sequence*), the *link* elements are not required.

The ordering of the list in the sequence provides the direction of flow (see Example A.8).
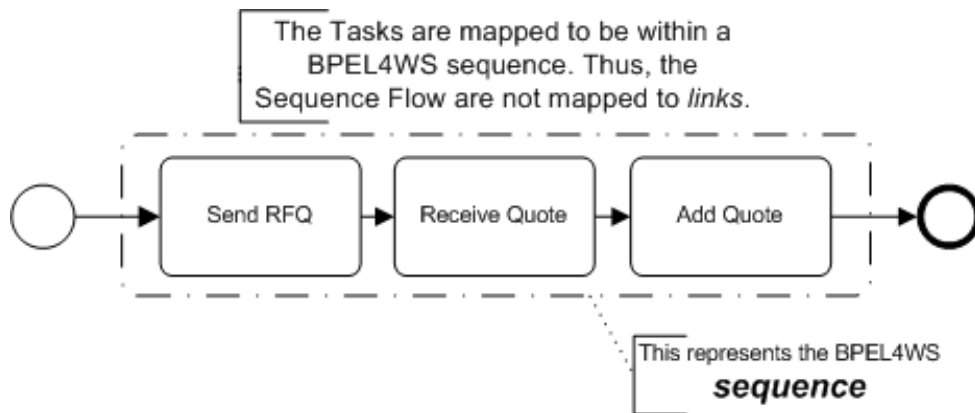
Figure A.8 - Example: Sequence Flow that are not used for BPEL4WS *links*

- *Link* elements are only appropriate when the Sequence Flow are Connecting Objects that are within a BPEL4WS *flow.*

However, it is only the Sequence Flow that are completely contained within the boundaries of the *flow* will be mapped to a *link* (see Example A.8). It should be noted that if another structured activity (e.g., a *switch*) is contained within the flow, then the Sequence Flow that would be appropriate for the contents of the structured activity, would not be mapped to a *link*.
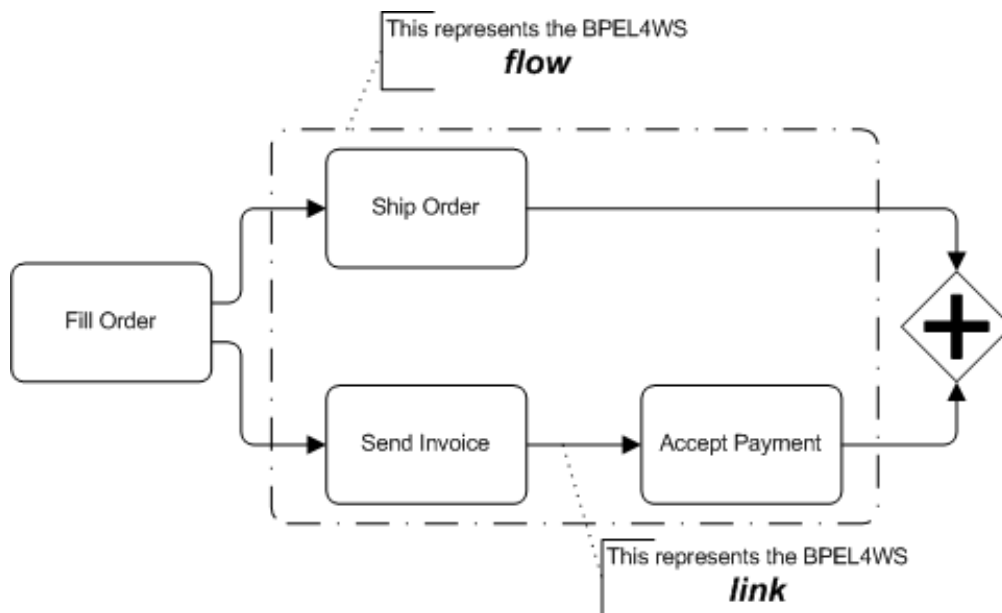
Figure A.9 - Example: A Sequence Flow that is used for a BPEL4WS *link*

A.1.11 Message Flow

A Message Flow does not have a specific mapping to a BPEL4WS element. It represents a message that is sent through a WSDL *operation* that is referenced in a BPEL4WS *receive*, *reply*, or *invoke*.

A.1.12 Association

An Association does not have a specific mapping to an execution language element. These objects and the Artifacts they connect to provide additional information for the reader of the BPMN Diagram, but do not directly affect the execution of the Process.

A.1.13 Exception Flow

BPMN Exception Flow is all the activities, connected by Sequence Flow, which flow from an Intermediate Event attached to the boundary of an activity, until the flow merges back into the Normal Flow (sometimes at the point of an End Event).

BPEL4WS handles exceptions in a much more structured and programmatic manner. If triggered through a *fault*, the activities in an *faultHandlers* will be performed and completed, and then the *process* will continue from the point where the interrupted activity would have completed normally. Thus, the *faultHandlers* element is a completely contained structured element.

Since BPMN handles Exception Flow with much more flexibility, so that the modeler can have the Exception Flow go wherever it is appropriate, there are different challenges to the BPEL4WS mapping, depending on the configuration of the BPMN model.

The following table displays the mapping Exception Flow to BPEL4WS:

Table A.43 - Common Exception Flow Mappings to BPEL4WS

Exception Flow	Mapping to BPEL4WS
Activities within the Exception Flow	All the activities that follow the attached Intermediate Event, until the Exception Flow merges back into the Normal Flow, will be mapped to BPEL4WS and then placed within the *faultHandlers* element for the *scope* of the activity (and usually within a *sequence*).

Additional BPEL4WS mapping patterns for Exception Flow will be described in the next three (3) sections.

The Exception Flow Merges back into the Normal Flow After the Activity

In this situation, the Exception Flow may contain one or more activities, but will merge back into the Normal Flow at the same object that follows the activity that is the source of the Exception Flow (see Figure A.10). This situation maps closely to the BPEL4WS mechanism for exception handling. Thus, no special mapping mechanisms are required.
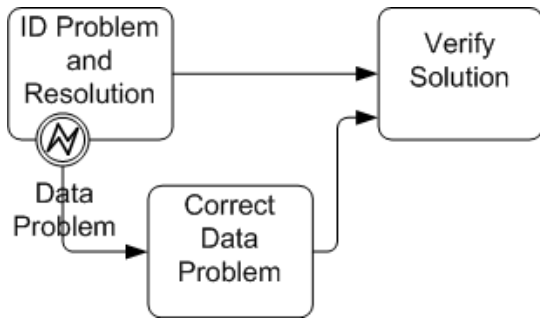
Figure A.10 - Exception Flow Merging back into Normal Flow Immediately after Interrupted Activity

The Exception Flow Merges back into the Normal Flow Further Downstream

In this situation, the activities in the Exception Flow substitute for some of the Normal Flow activities and, thus, the Exception Flow will skip these activities and merge into the Normal Flow further downstream (see Figure A.11). Alternatively, the exception may create a situation where the Process must end prematurely, which means that the Exception Flow will merge with the Normal Flow at an End Event (see Figure A.12). In either situation, special BPEL4WS patterns will have to appended to the basic Exception Flow mappings.
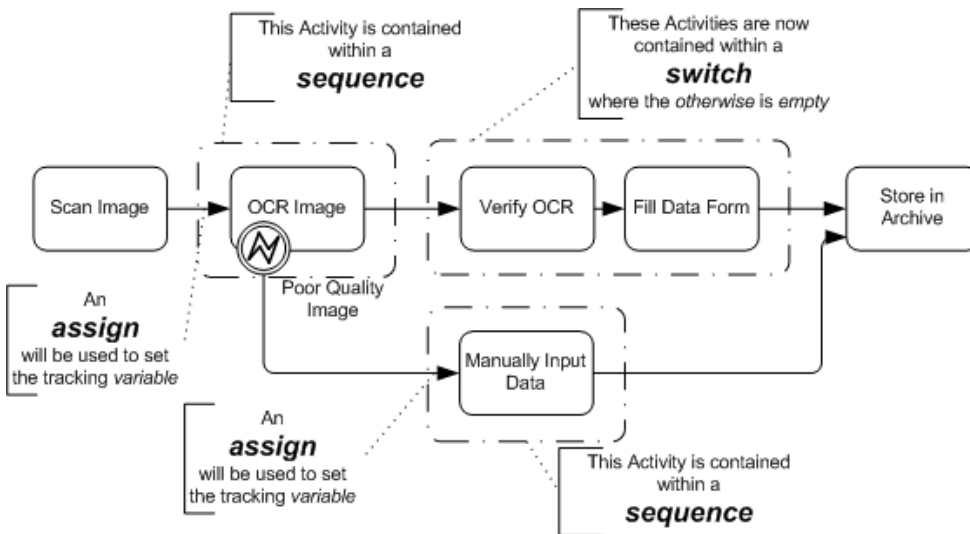
Figure A.11 - Exception Flow Merging back into the Normal Flow Further Downstream

The following table displays the mapping Exception Flow to BPEL4WS (these mappings extend the mappings common to Exception Flow -- see above):

Table A.44 - Exception Flow Merging back into the Normal Flow Further Downstream

Exception Flow	Mapping to BPEL4WS
Activities within the Exception Flow	If there is only one activity in the *faultHandlers* element for the *scope* of the activity, then this activity will be placed within a *sequence* and preceded by an *assign* (as described below).
Original Activity	The mapping of the original activity will be placed within a *sequence* (if it had not bee already).
After the Original Activity	The original activity will now be followed by a *switch*, instead of what would have been normally mapped there.
Switch Characteristics	The *switch* will be binary in nature. There will be one *case* and an *otherwise* element.

Create the tracking variable	A *variable* must be used so that the switch will know whether or not the Exception Flow or Normal Flow had reached that point in the Process. To do this, a BPEL4WS *variable* must be created with a derived name and will have a structure as follows:
	
	```<variable name="[activty.Name]_normalCompletion"```
	``` messageType="noDefaultRequired" />```

Supporting WSDL Message	A WSDL *message* element will have to be created to support this *variable*. This *message* can be used for multiple *variables*. The *message* will be structured as follows:
	
	```<message name="noDefaultRequired" >```
	``` <part name="normalCompletion" type="xsd:boolean" />```
	```</message>```

Initialization of the Tracking Variable	An *assign* activity will be created to initialize the *variable* before the start of the original activity. It will be the first activity in the *sequence* described above. The *assign* will be structured as follows:
	
	```<assign name="[activity.Name]_initialize_normalCompletion">```
	``` <copy>```
	``` <from expression="true"/>```
	``` <to variable="[activity.Name]_normalCompletion"```
	``` part="normalCompletion" />```
	``` </copy>```
	```</assign>```

Setting of the tracking variable	If a fault is thrown and *faultHandlers* is activated, then an *assign* activity will be used to set the *variable* to False. This will be the first activity within the *sequence* activity of the *faultHandlers*. The *assign* will be structured as follows:
	
	```<assign name="[activity.Name]_set_normalCompletion">```
	``` <copy>```
	``` <from expression="false"/>```
	``` <to variable="[activity.Name]_normalCompletion"```
	``` part="normalCompletion" />```
	``` </copy>```
	```</assign>```

Table A.44 - Exception Flow Merging back into the Normal Flow Further Downstream

Exception Flow	Mapping to BPEL4WS

Switch cases	The case for the switch will contain all the mappings for all activities that occur in the Process until the Exception Flow has merged back (which could be the end of the Process), usually within a *sequence*. The otherwise for the switch will contain an *empty* activity.
	
	The condition for the *switch case* will used the `normalCompletion` *variable* and will structured as follows:
	
	``
	<switch>
	  <case condition="bpws:getVariableProperty(
	        [activity.Name]_normalCompletion,
	        normalCompletion)=true">
	    <sequence>
	
	<!--The mappings of the Process activities until the merging
	   of the Exception Flow are placed here.-->
	
	    </sequence>
	  </case>
	  <otherwise>
	    <empty/>
	  </otherwise>
	</switch>
	``

Potential Invalid Model	If the Exception Flow occurs in the larger context of a set of parallel activities, then the Exception Flow must merge back into the Normal Flow prior to the end of the parallel activities (a BPEL4WS *flow*), or this will create an invalid model.
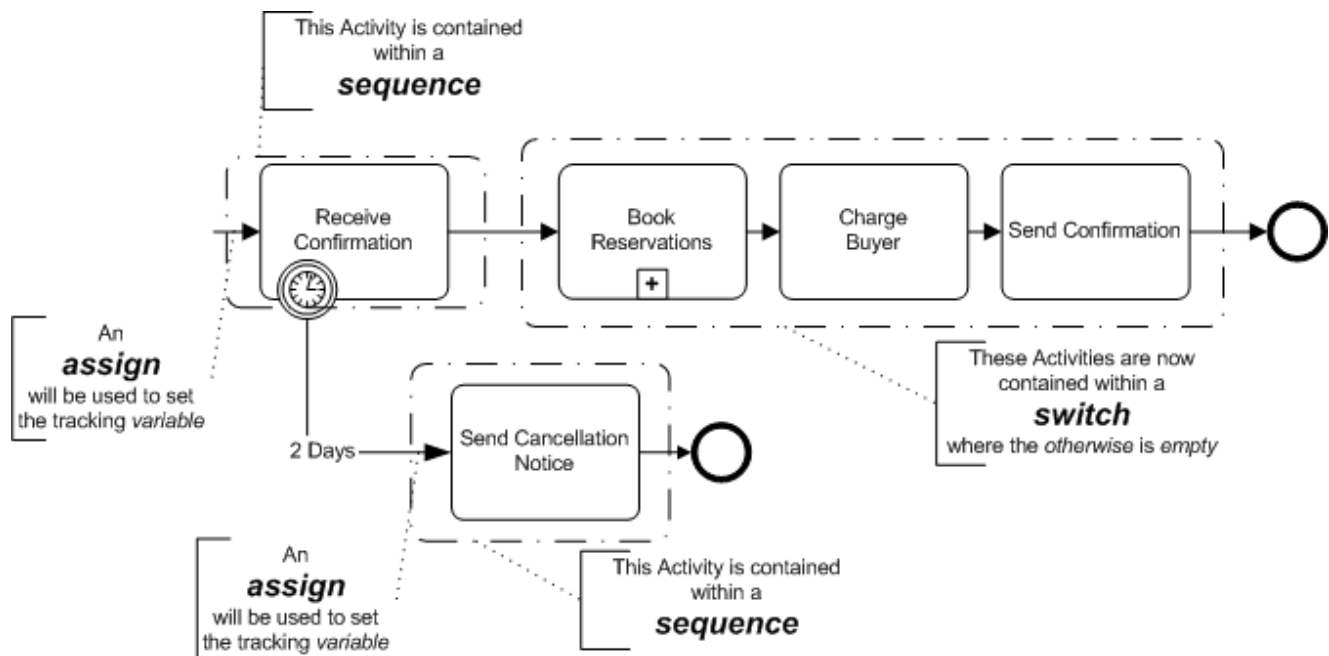
Figure A.12 - Exception Flow Merging back into the Normal Flow at the End Event

The Exception Flow Loops back into the Normal Flow Upstream

In this situation, the Exception Flow will loop back into the Normal Flow prior to the completion of the activity that is the source of the Exception Flow (see Figure A.13). This is a particularly challenging mapping and cannot be done entirely within the confines of the original BPEL4WS *process*. Another process will need to be derived and then "spawned" until the original activity can be completed normally.
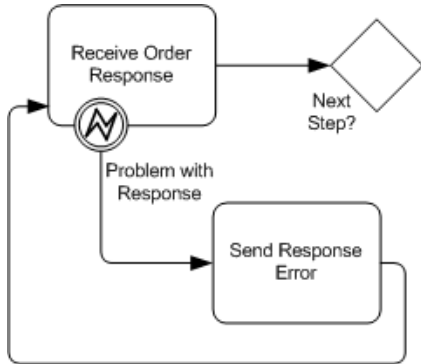
Figure A.13 - Example of Exception Flow Looping Back into the Normal Flow Upstream

This part of the Process will be modified at the BPEL4WS level so that the loop can be performed (through calling another *process*). If the flow moves to the *faultHandlers* activity, this means that the original activity will need to be performed again. Thus, the original activity will be duplicated in another *process* and the *faultHandlers* will contain a one-way *invoke* to "spawn" this other process (see Figure A.14). In addition, the original process will wait with a *receive* activity for a message from the derived process that the original activity has completed normally.
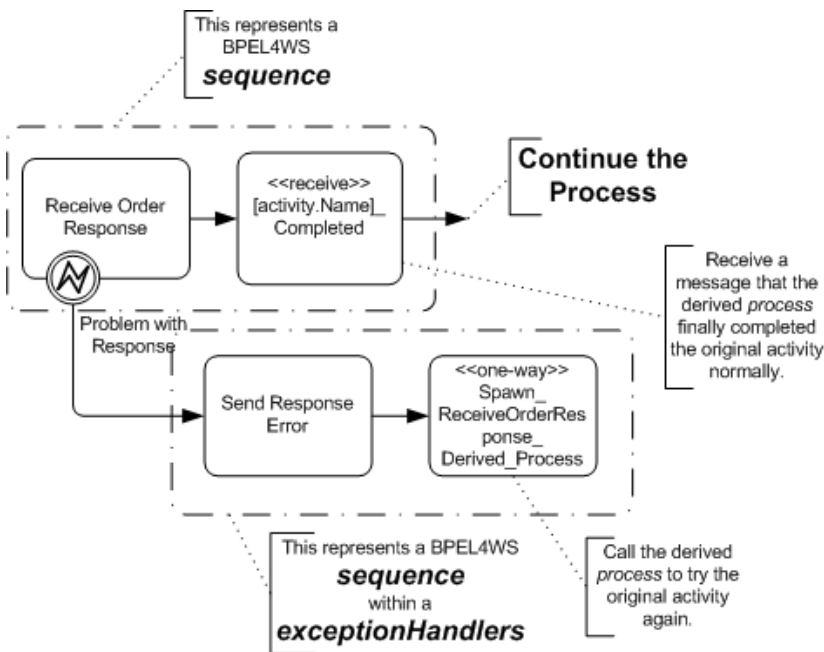
Figure A.14 - Example of Modification at BPEL4WS level to Handle the Loop

The derived process will be structure much like the corresponding section of the original process (see Figure A.15). The mappings of the original activities, from the point of the BPMN Process where the Exception Flow loops into the Normal Flow to the point of the source of the Exception Flow, will be in the derived *process*. The same faultHandlers will be attached to the scope around the original activity. The *faultHandlers* will also contain a one-way *invoke* to "spawn" itself if the fault occurs again.

When the original activity finally completes normally, one-way *invoke* will be used to send a message back to the original *process* so that normal activities can continue.
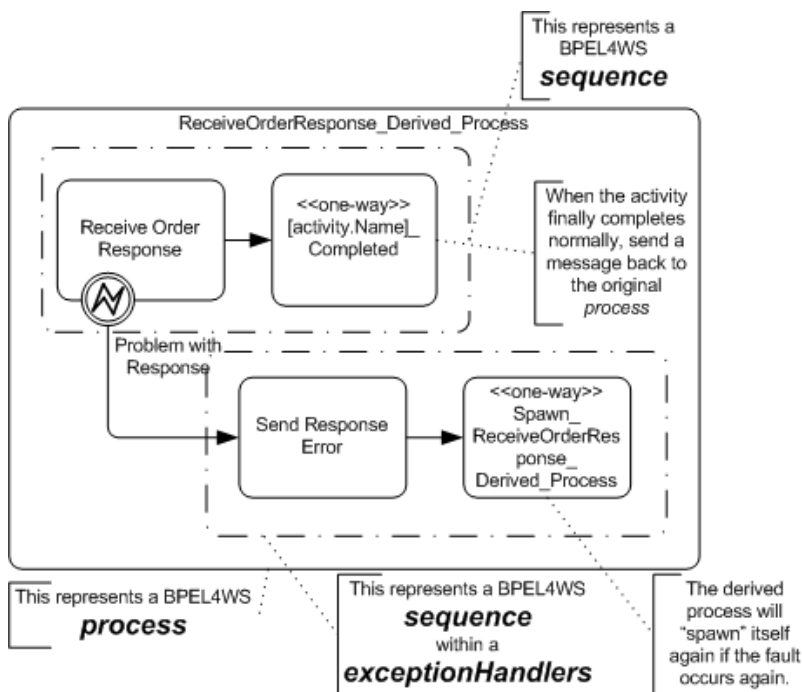
Figure A.15 - Example of a Derived Process to Handle the Looping

A.1.14 Compensation Association

The following table displays a set of mappings from a Compensation Association to BPEL4WS elements:

Table A.45 - Exception Flow Mappings to BPEL4WS

Compensation Association	Mapping to BPEL4WS
A Compensation Intermediate Event attached to an activity boundary	The mapping of the Compensation Event is described in "Compensation Intermediate Events" on page 153. The mapping of the activity Associated with the Intermediate Event will follow the mapping rules defined in Section , "Task Mappings," on page 175 or in Section , "Sub-Process Mappings," on page 173 will be placed within the *compensationHandler* element.

A.1.15 Assignment Mapping

The following table displays a set of mappings from the variations of an Assignment expression to BPEL4WS elements:

Table A.46 - Assignment Mappings to BPEL4WS

Assignment	Mapping to BPEL4WS

To	The To attribute will map to the *to* element of the BPEL4WS *assign* activity. A variable and supporting WSDL message should have already be created for the Property used in for the Assignment To attribute. Thus, the structure of the to element will be as follows: If the Property is an attribute of a Process:
	`<to variable="[Process.Name]_ProcessData"`
	`    part="[Property.Name]" />`
	
	If the Property is an attribute of an activity:
	`<to variable="[activity.Name]_ActivityData"`
	`    part="[Property.Name]" />`

From	The From expression will map to the *from* element of the BPEL4WS *assign* activity. `<from expression="[From Expression]" />`

A.1.16 BPMN Supporting Type Elements

This section describes the mapping to BPEL4WS of a non-graphical elements that are part of BPMN. Messages, which are linked with Message Flow, do have impact on how many other BPMN elements are mapped to BPEL4WS.

Messages

The following are the mappings of a Message. These mappings are used to create a BPEL4WSE4WS XML file, plus a supporting WSDL supporting file. These mappings are used for a Start Event, End Event, Intermediate Event, and Task:

Table A.47 - Message Attributes

Attributes	Description
Name	The Name attribute maps to the *name* attribute of a BPEL4WS *variable* element. Note that the extra spaces and non-alphanumeric characters MUST be stripped from the Name to fit with the XML specification of the *name* attribute. Note that there may be two or more elements with the same name after the BPMN name has been stripped. The *messageType* attribute of the *variable* element refers to a WSDL *message* type definition. Thus, the *messageType* will share the same Name and a corresponding WSDL *message* must be created.
Properties	Each Properties of the BPMN Message will map to a *part* element of the WSDL *message*. The Name attribute of the Property will map to the *name* attribute of the *part*. The Type attribute of the Property will map to the *type* attribute of the *part*.

A.1.17 Determining the Extent of a BPEL4WS Structured Element

The structure and vocabulary of BPMN differs from BPEL4WS. BPMN allows flexible, and free-form methods of connecting activities through Sequence Flow. Furthermore, BPMN is cyclical in that it allows Sequence Flow to connect to upstream objects so that a modeler can easily visualize looping situations. BPEL4WS has a much more structured form

of creating a process flow. The *flow* activity in BPEL4WS does allow some flexibility with its *link* elements, but is acyclical. Thus, there is not going to be a one-to-one mapping of the BPMN elements to the BPEL4WS elements, without restricting the connection capability of BPMN.

This is particularly true of the BPEL4WS. In BPEL4WS, structure elements, such as *switch*, *pick*, and *while,* have a clear beginning and end. BPMN does not provide specific markers for the start and end of these elements. The exact configuration of the Sequence Flow connections will determine how the Process will be mapped to the BPEL4WS elements.

To determine the appropriate merging and joining points that are needed to construct the structured elements, the configuration of the Process needs to be analyzed. The mechanism we are proposing is called Token Analysis. This involves the creation of a conceptual Token that will "traverse" all the Sequence Flow of the Process. The Token will have a hierarchical TokenId set that will expand/or contract based on the forking and joining and/or splitting and merging that occurs throughout the Process. By matching the TokenId set of Tokens that arrive at objects that have multiple incoming Sequence Flow, it will be possible to determine the boundaries of execution language structured activities.

A BPMN Gateway will usually indicate the start of a BPEL4WS structured element, but even this may not be one-to-one if there are loops involved. The end of the BPEL4WS structured element is even less obvious, since it could be marked by the convergence of Sequence Flow into most types of BPMN elements.

The following sections will describe how different BPMN configurations will map to the BPEL4WS structure elements and show how conceptual Tokens can be used to determine the extent of the BPEL4WS elements.

Identifying the Start of a BPEL4WS Element

The most basic structured element of BPEL4WS is the sequence.

u   If the *process*, or the activity of a structured element (e.g., a *switch case*), contains more than one activity, then it is likely a *sequence* will be needed. Nearly any set of activities connected by Sequence Flow, which is not going to be mapped to the contents of a *flow*, will be contained within a *sequence*. The *sequence* will envelope all the remaining elements to the extent of the context in which the *sequence* exists. For example, the *sequence* will extend the length of the *process*, or the length of a *switch case*, etc.

For the other types of BPEL4WS elements, their extend is determined by tracing through the Process with conceptual Tokens:

u   First the start of the BPEL4WSE4WS structured element (e.g., *flow*, *switch*, *pick*, etc.) must be identified. This is done by performing the mapping of the BPMN elements, starting with the Start Event or first element(s) if there is no Start Event, and proceeding down the Sequence Flow. The start of the structured element is usually a Gateway or if an activity has multiple outgoing Sequence Flow (see Figure A.16 and Figure A.18).

u   Note that some structured elements (mainly a *sequence*, but including others such as a *switch*) are needed for mapping a particular BPMN activity (as described in the sections above). In these cases, the extent of these structured elements are known.
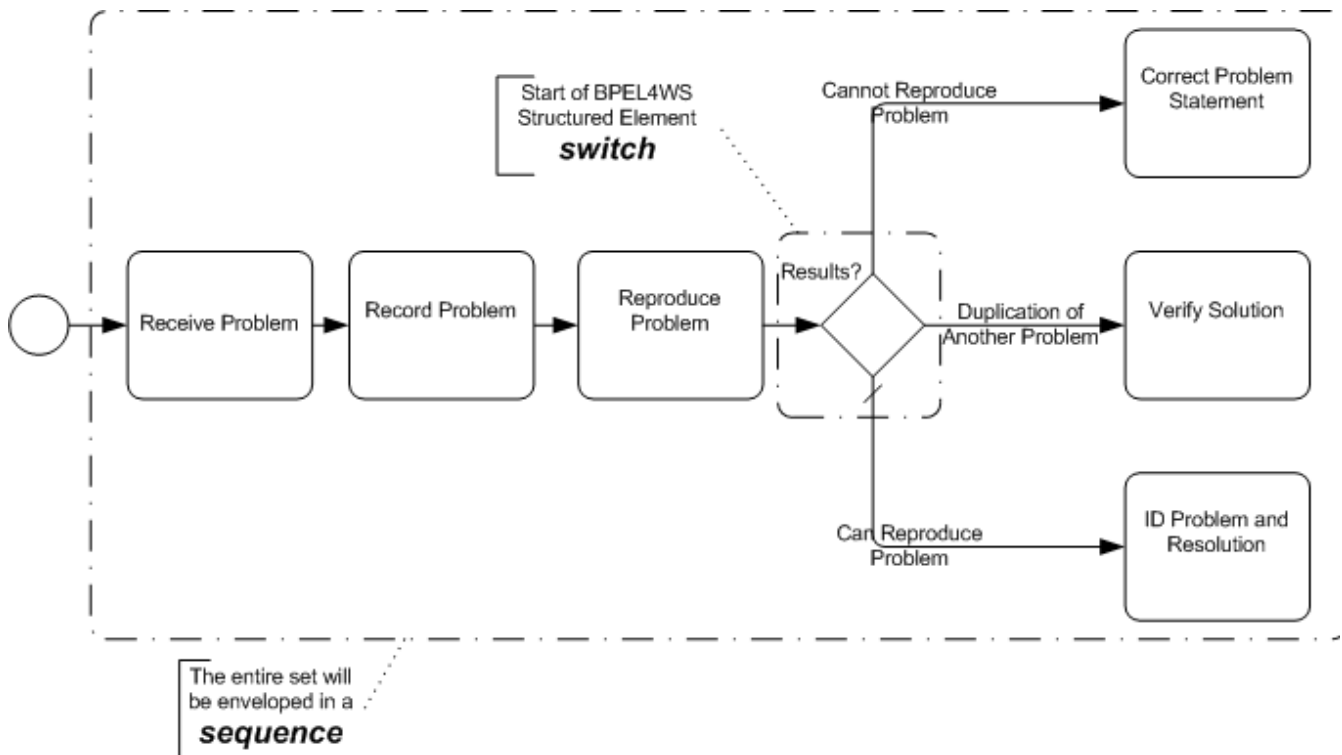
Figure A.16 - Identification of BPEL4WS structured element

u   The number paths that make up the structured element MUST be determined. To do this, the all outgoing paths from the location of the structured element will be identified. A conceptual Token can be used to trace the paths. The Tokens are given an Id that uniquely identifies the precedent of the structure element being determined and the number of paths being traced for that element (see Figure A.17).
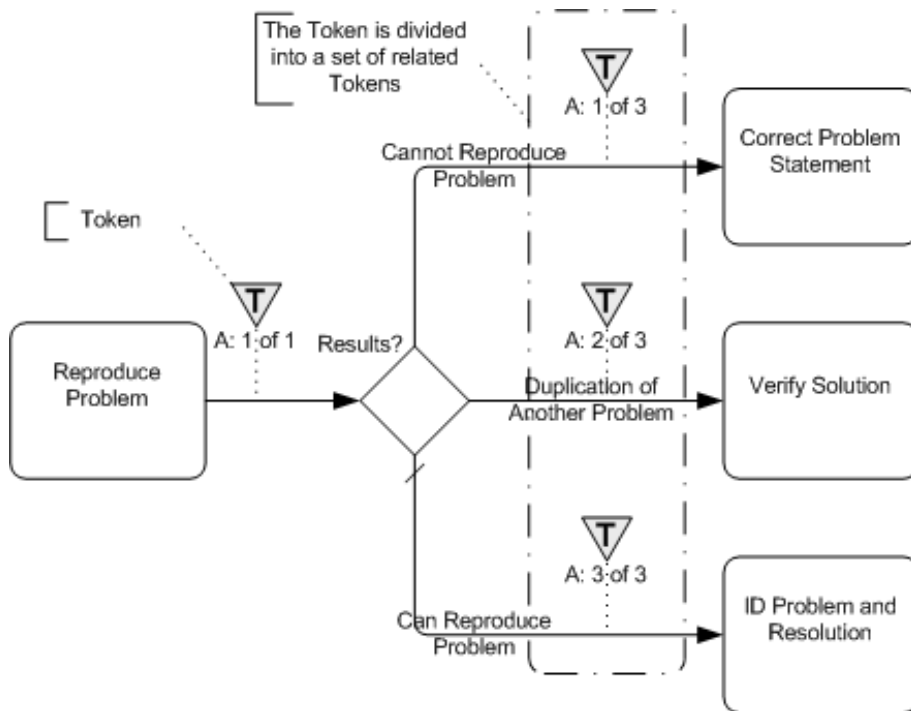
Figure A.17 - The Creation of Related Tokens

Finding the End of a BPEL4WS Element

The end of a BPEL4WS structured element will be found when all the paths, which were identified at the start of the element, have converged.

u   Trace each path until there is a merge or join with <u>all</u> the other paths. When all the Tokens with the appropriate Ids arrive at the same BPMN object and can be recombined, then the structured element SHALL be closed (see Figure A.18).
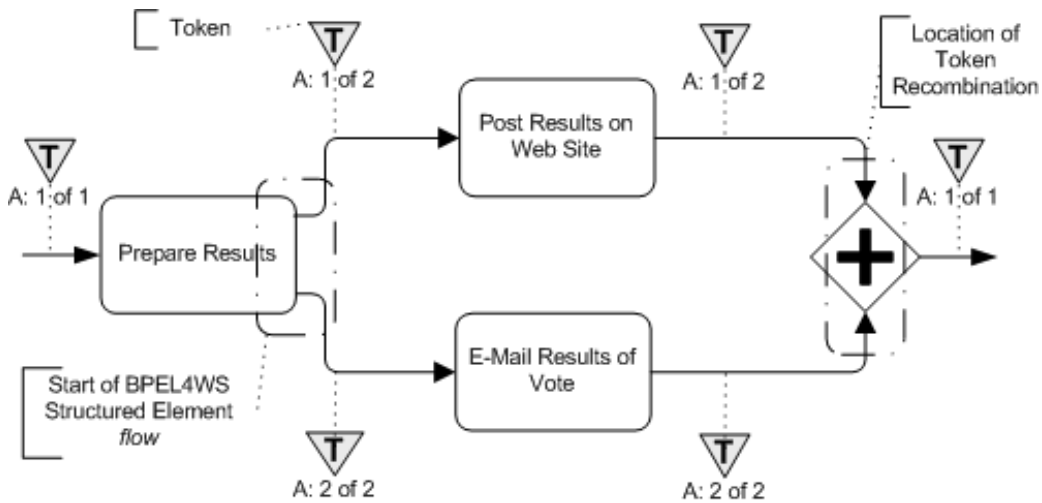
Figure A.18 - Example of Recombination of Tokens

u　There MAY be partial recombinations of the Tokens prior to the final recombination. In this case, one Token will contain all the identities of the Tokens that have been merged (see Figure A.19). Note that partial recombination of a Token creates another mapping issue that is described in Section , "BPMN Elements that Span Multiple BPEL4WS Sub-Elements," on page 207.
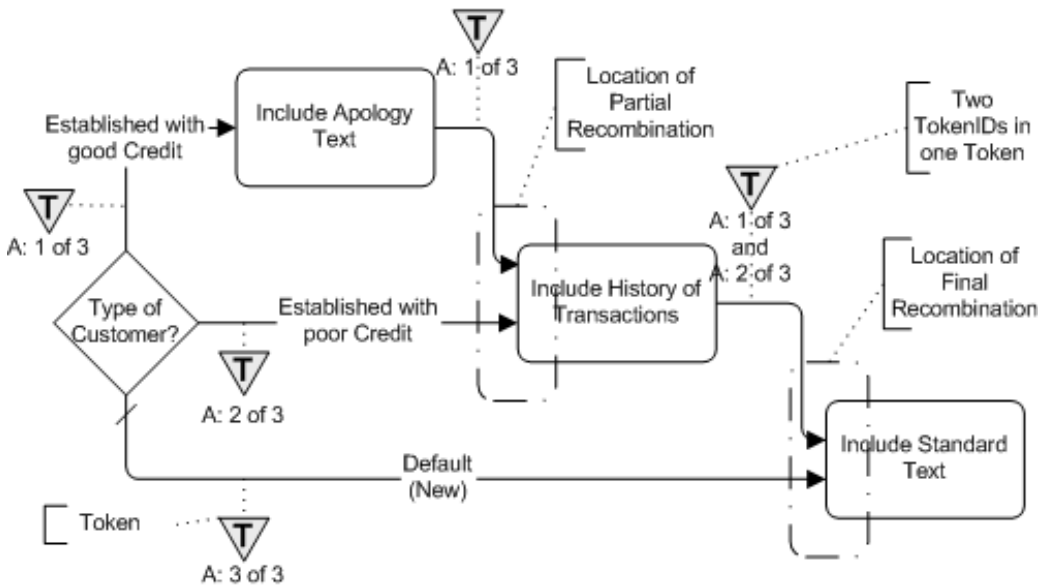
Figure A.19 - Example of Partial Recombination of Tokens

u　End Events can be combined with other BPMN objects to complete the merging or joining of the paths of a BPEL4WS structured element (see Figure A.20).
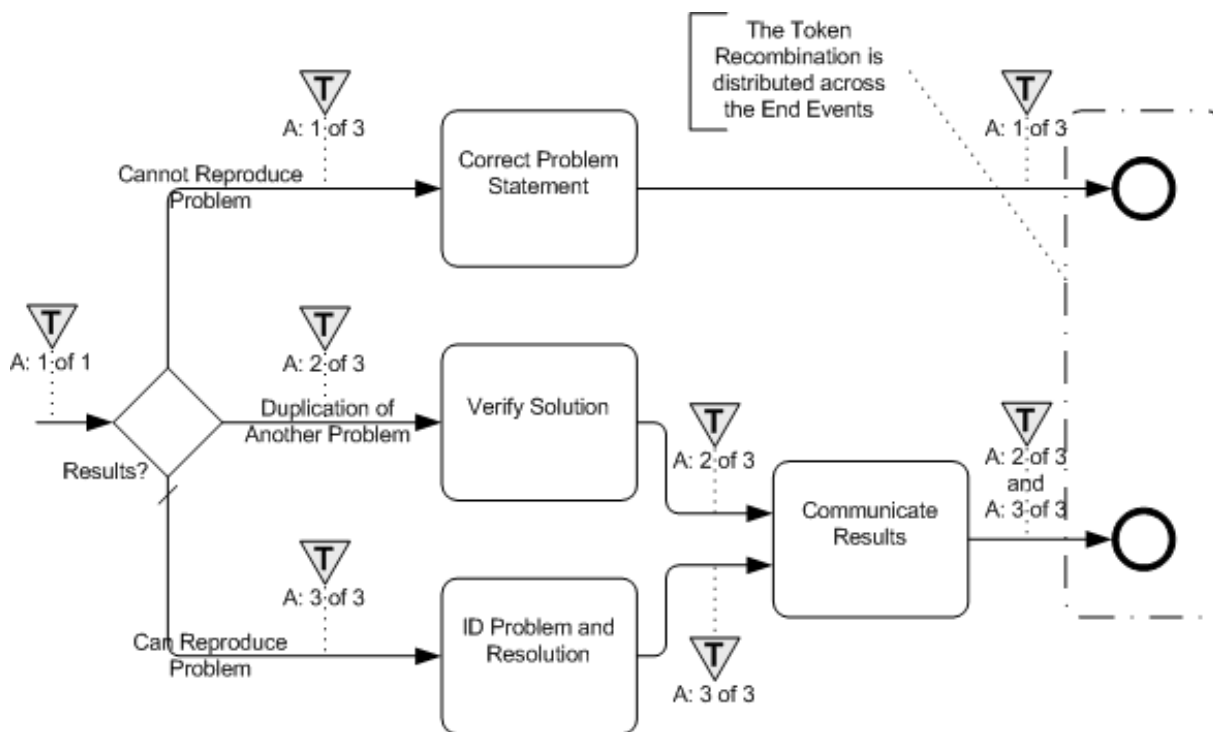
Figure A.20 - Example of Distributed Token Recombination

Nested Elements

Another structured element may occur before the first structure element is closed.

  u   If another structured element is encountered before all the paths are merged (see Figure A.21), then the tracing of the first element MUST be stopped and the tracing of the paths of the second element MUST begin. The extent of the second element MUST be determined before the extent of the first element can be determined.

  u   This process MUST be repeated if other structured elements are encountered during the tracing of any paths of structured elements.
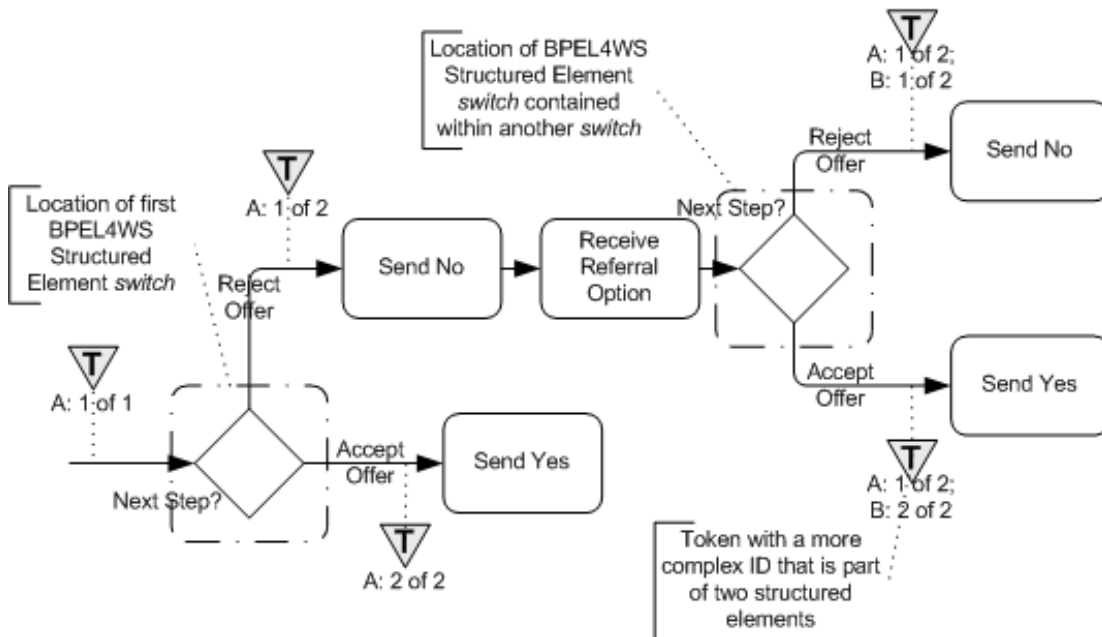
Figure A.21 - Example of nested BPEL4WS structural elements

Handling Loops

Loops are created when the flow of the Process moves from a downstream object to an upstream object.

u   If one of the paths arrives at a BPMN object that is upstream from the source of the structured element, then this SHALL create a looping situation. How the loop is handled depends on the type structured element is being traced and how many paths are included in the element.

The following sections will describe the mapping for the different type of loop configurations.

Simple Loop From a Gateway

This type of loop is created by a Gateway that has only two outgoing Sequence Flow. One Sequence Flow continues downstream and the other loops back upstream (see Figure A.22). Note that there might be intervening activities prior to when the Sequence Flow loops back upstream.

u   This will map to a BPEL4WS *while* activity.

   u   The Condition for the Sequence Flow that loops back upstream will map to the *condition* of the *while*.

   u   All the activities that span the distance between where the loop starts and where it ends, will be mapped and placed within the activity for the *while*, usually within a *sequence*.
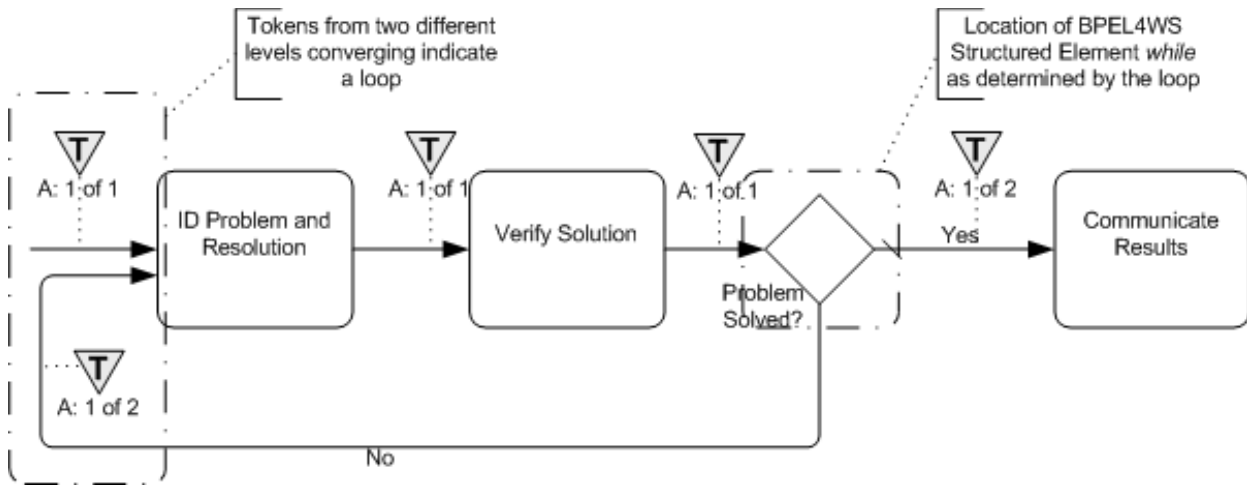
Figure A.22 - Example of a Loop from a Decision with Two Alternative Paths

Loop/Switch Combinations From a Gateway

This type of loop is created by a Gateway that has three or more outgoing Sequence Flow. One Sequence Flow loops back upstream while the others continue downstream (see Figure A.23). Note that there might be intervening activities prior to when the Sequence Flow loops back upstream.

u This maps to both a BPEL4WS *while* and a *switch*. Both activities will be placed within a *sequence*, with the *while* preceding the *switch*.

   u For the *while*:

   u The Condition for the Sequence Flow that loops back upstream will map to the *condition* of the *while*.

   u All the activities that span the distance between where the loop starts and where it ends, will be mapped and placed within the activity for the *while*, usually within a *sequence*.

   u For the *switch*:

   u For each additional outgoing Sequence Flow there will be a *case* for the *switch*. The details for mapping to a switch from a Gateway can be found in Section A.1.6, "Gateways," on page 178.
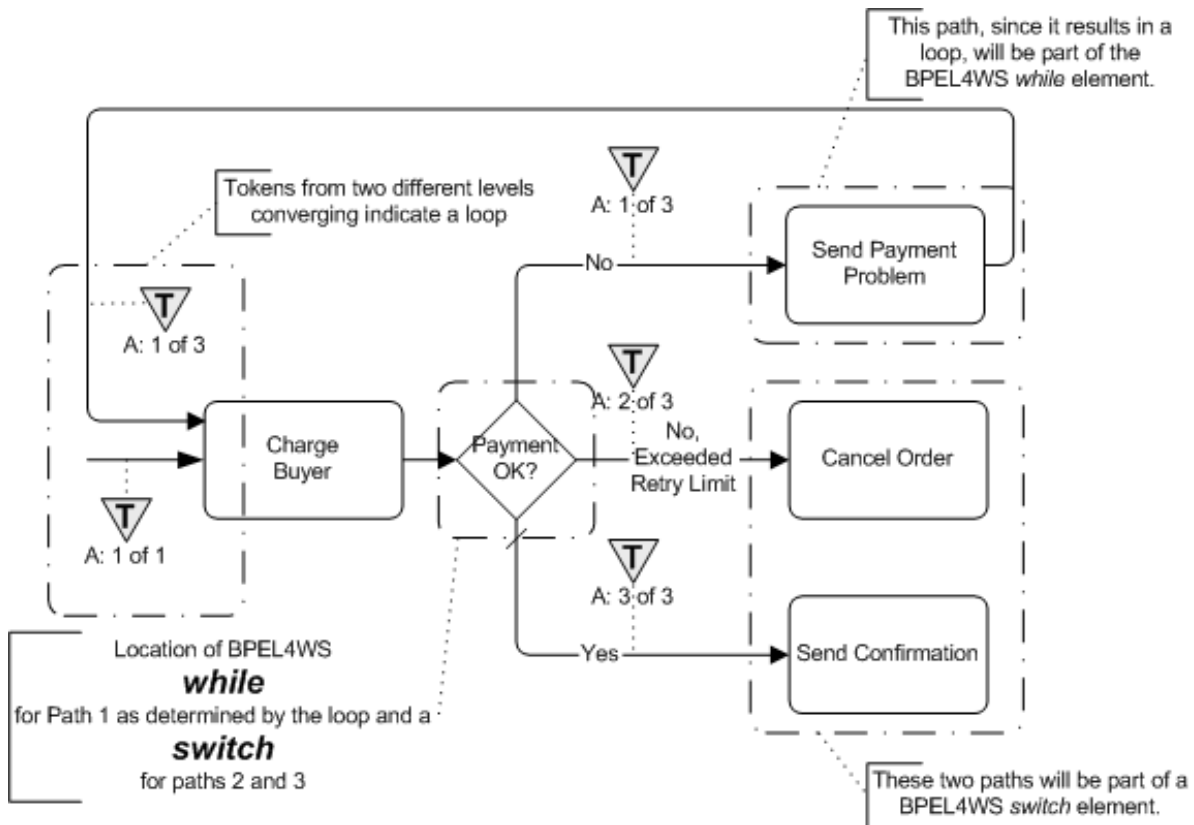
Figure A.23 - Example of a Loop from a Decision with more than Two Alternative Paths

Interleaved Loops

This is a situation where there at least two loops involved and they are not nested (see Figure A.24). Multiple looping situations can map, as described above, if they are in a sequence or are fully nested (e.g., one *while* inside another *while*). However, if the loops overlap in a non-nested fashion, as shown in Figure A.24, then the structured element *while* cannot be used to handle the situation. Also, since a *flow* is acyclic, it cannot handle the behavior either.
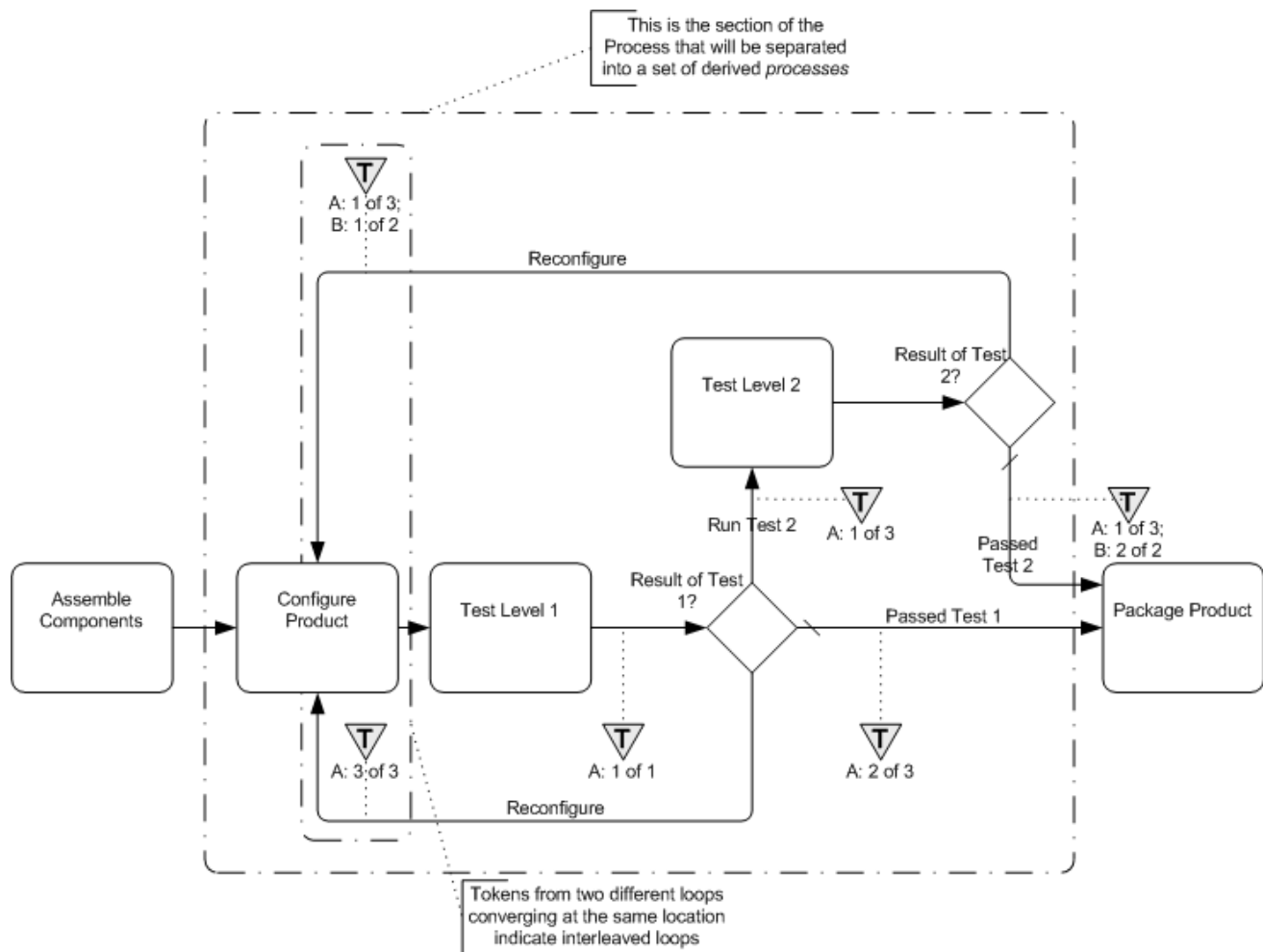
Figure A.24 - Example of Interleaved Loops

To handle this type of behavior, parts of the BPEL4WS *process* will have to be separated into one or more derived *processes* that are spawned from the main *process* and will also spawn or call each other (note that the examples below are using a spawning technique). Through this mechanism, the linear and structured elements of BPEL4WS can provide the same behavior that is shown through a set of cycles in a single BPMN diagram. To do this:

u   The looping section of the process, where the loops first merge back (upstream) into the flow until all the paths have merged back to Normal Flow, shall be separated from the main *process* into a set of derived *processes* that will spawn each other until all the looping conditions are satisfied.

   u   The section of the Process that is removed will be replaced by a (one-way) *invoke* to spawn the derived *process*, followed by a *receive* to accept the message that the looping sections have completed and the main *process* can continue (see Figure A.25).

   u The name of the *invoke* will be in the form of:

      u   "Spawn_[(loop target)activity.Name]_Derived_Process"

   u The name of the *receive* will be in the form of:

      u   "[(loop target)activity.Name]_Derived_Process_Completed"
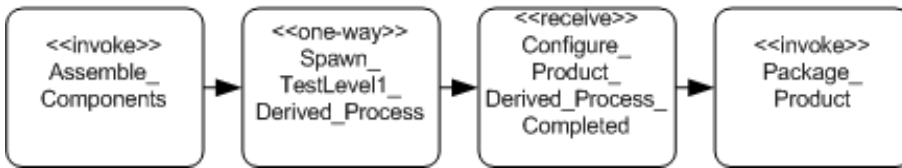
Figure A.25 - Example of the BPEL4WS Pattern for Substituting for the Derived *Process*

u   For each location in the Process where a Sequence Flow connects upstream, there will be a separate derived BPEL4WS *process*.

   u   The name of the derived process will be in the form of:

      u "[(loop target)activity.Name]_Derived_Process"

   u   All Gateways in this section will be mapped to *switch* elements, instead of *while* elements (see Figure A.26).

   u   Each time there is a Sequence Flow that loops back upstream, the activity for the *switch case* will be a (one-way) *invoke* that will spawn the appropriate derived *process*, even if the *invoke* spawns the same *process* again.

      u The name of the *invoke* will the same as the one describe above.

   u   At the end of the derived *process* a (one-way) *invoke* will be used to signal the main process that all the derived activity has completed and the main *process* can continue.

      u The name of the *invoke* will be in the form of:

         u   "[(loop target)activity.Name]_Derived_Process_Completed"
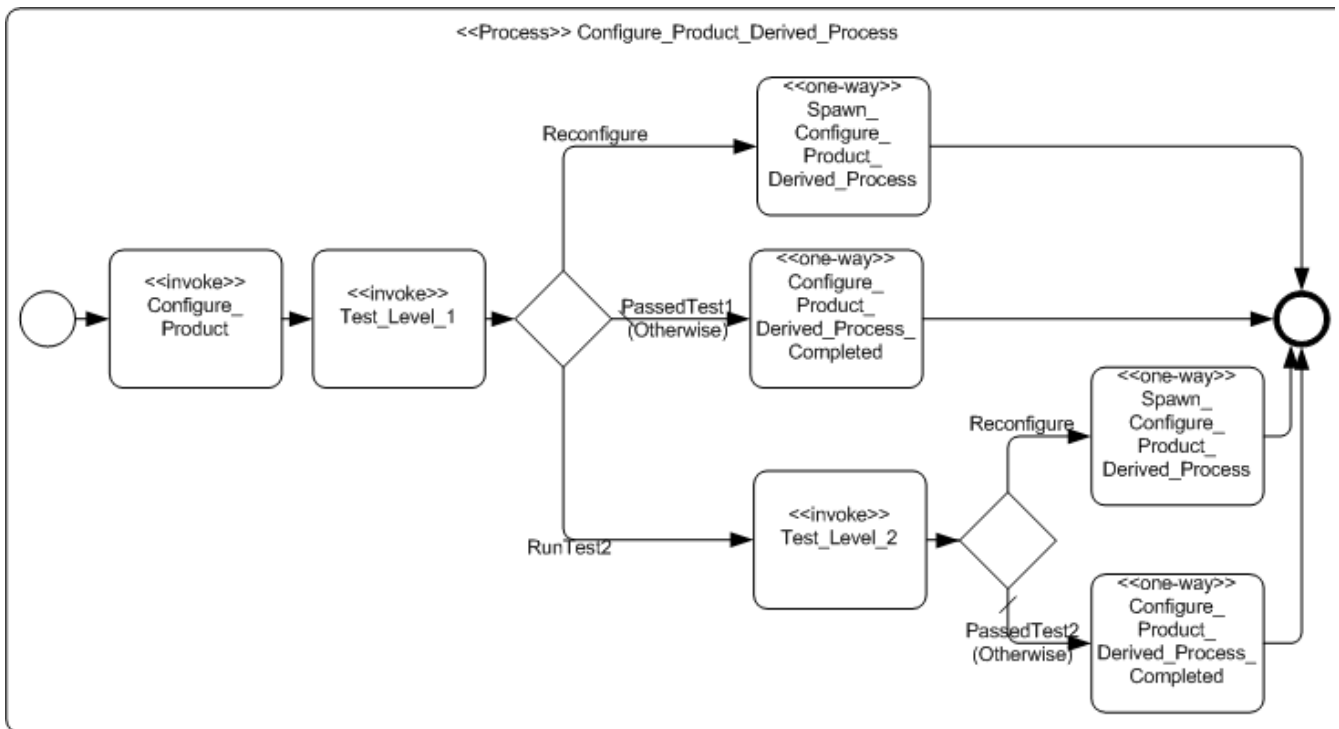
Figure A.26 - Example of a BPEL4WS Pattern for the Derived Process

Infinite Loops

This type of loop is created by a Sequence Flow that loops back without an intervening Gateway to create alternative paths (see Figure A.27). While this may be a modeling error most of the time, there may be situations where this type of loop is desired, especially if it is placed within a larger activity that will eventually be interrupted.

- u This will map to a *while* activity.
   - u The condition of the while will be set to an expression that will never evaluate to True, such as condition "1 = 0."
   - u All the activities that span the distance between where the loop starts and where it ends, will be mapped and placed within the activity for the *while*, usually within a *sequence*.
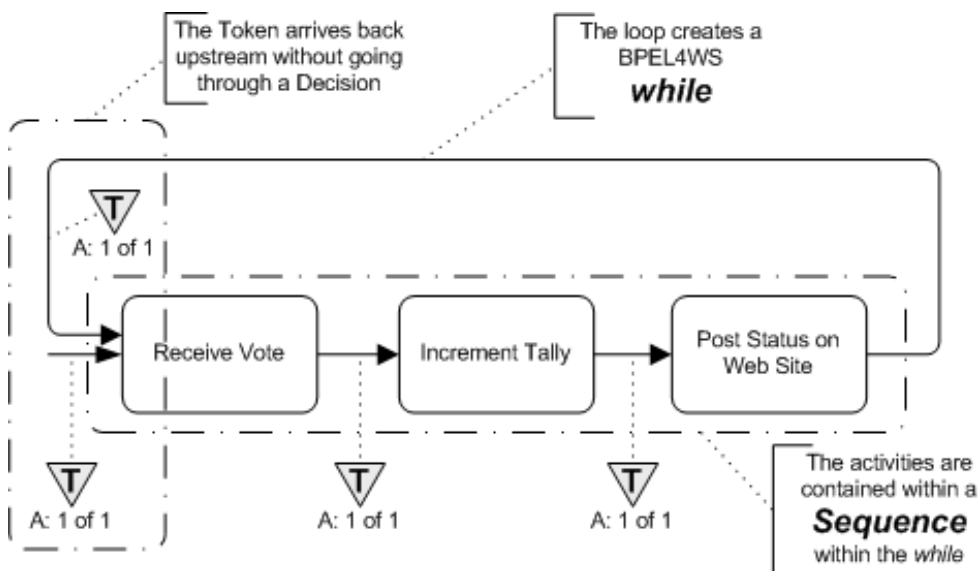
Figure A.27 - Example: An Infinite Loop

Handling Link Events as Go To Objects

As was seen in Figure 10.43, Figure 10.44, and Figure 10.45, Link Intermediate Events can be used as Go To Objects. The basic impact of using them in such a way is that they are a substitute using a single, longer Sequence Flow to make the same connection between two objects. Thus, the mapping to BPEL4WS should be done by considering them as just a single Sequence Flow. This means that the Intermediate Events are not mapped to any BPEL4WS element. Instead a conceptual Sequence Flow will be used, with the Source and Target of that Sequence Flow being the Source of the Sequence Flow going into the Source Link Event and the Target of the Sequence Flow coming out of the Target Link Event (see Figure A.28). The mapping at this point can done using all the mapping consideration described in this Chapter.
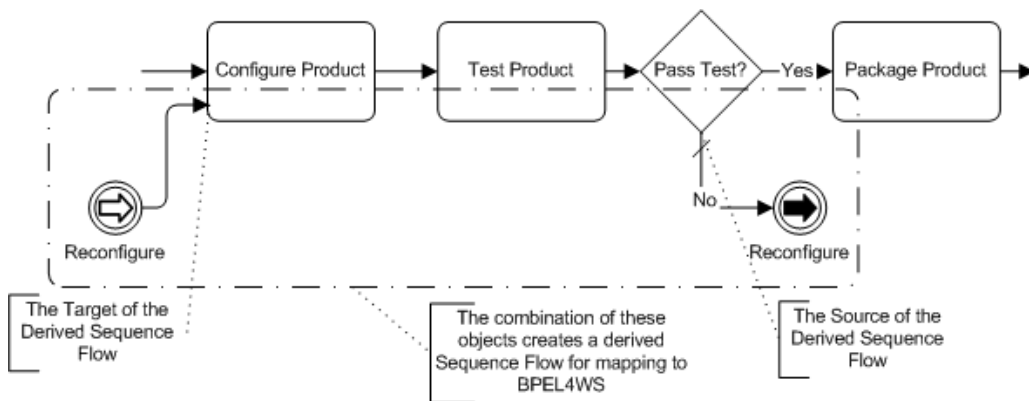
Figure A.28 - Example: A Pair of Go To Link Events are Treated as a Single Sequence Flow

BPMN Elements that Span Multiple BPEL4WS Sub-Elements

Figure A.19 is repeated below in Figure A.29 to illustrate how BPMN objects may exist in two separate sub-elements of a BPEL4WS structured element at the same time. Since BPMN allows free form connections of activities and Sequence Flow, it is possible that two (or more) Sequence Flow will merge before all the Sequence Flow that map to a BPEL4WS structure element have merged. The sub-elements of a BPEL4WS structured elements are also self contained and there is no cross sub-element flow. For example, the *cases* of a *switch* cannot interact; that is, they cannot share activities. Thus, one BPMN activity will need to appear in two (or more) BPEL4WS structured elements.

There are two possible mechanisms to deal with the situation.

- u First, the activities are simply duplicated in all appropriate BPEL4WS elements.
- u Second, the activities that need to be duplicated can be removed from the main process and placed in a derived process that is called (*invoked*) from all locations in the BPEL4WS elements as required.
  - u The name of the derived process will be in the form of:
    - u "[(target)object.Name]_Derived_Process"

In Figure A.29 displays this issue with an example. In that example, two Sequence Flow merge into the "Include History of Transactions" Task. However, the Decision that precedes the Task has three (3) alternatives. Thus, the Decision maps to a BPEL4WS *switch* with three (3) *cases*. The three cases are not closed until the "Include Standard Text" Task, downstream. This means that the "Include History of Transactions" Task will actually appear in two (2) of the three (3) *cases* of the *switch*.

Note – the use of a BPEL4WS *flow* will be able to handle the behavior without duplicating activities, but a *flow* will not always be available for use in these situations, particularly if a BPEL4WS *pick* is required.
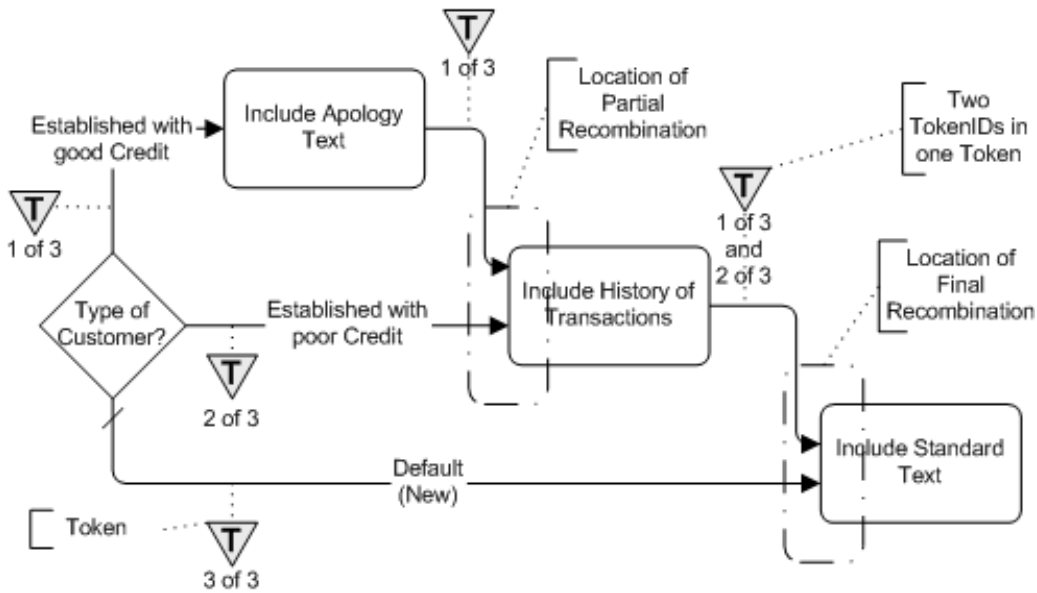
Figure A.29 - **Example: Activity that spans two paths of a BPEL4WS Structured Element**

Example A.8 displays some sample BPEL4WS code that reflects the portion of the Process that was just discussed and is shown in Figure A.29. Note that there are two *invoke* elements that have the same *name* attribute ("IncludeHistoryofTransactions").

```
<!--Continue with the process-->
<switch name="TypeofCustomer">
  <!-- name="Established with Good Credit" -->
  <case condition="bpws:getVariableProperty(ProcessData,CreditType)>"Yes, Good"">
    <invoke name="IncludeApologyText" ...>
    <!--This also exists in the other case-->
    <invoke name="IncludeHistoryofTransactions" ...>
  </case>
  <!--name="Established with poor Credit" -->
  <case condition="bpws:getVariableProperty(ProcessData,CreditType)>"Yes, Poor"">
    <!--This also exists in the other case-->
    <invoke name="IncludeHistoryofTransactions" ...>
  </case>
  <!--name="Default (New)" -->
  <otherwise>
    <!--Nothing happens here-->
    <empty/>
  </otherwise>
</switch>
<invoke name="IncludeStandardText" ...>
<!--Continue with the process-->
```

Example A.8 - Example: BPMN Elements that Span Multiple BPEL4WS Sub-Elements

A.2 BPMN by Example (Including a Mapping to BPEL4WS)

This section will provide an example of a business process modeled with BPMN and will extend Section 11 by adding information about how the example Process will map to BPEL4WS. The process that will be described is a process used to help develop this notation. It is a process for resolving issues through e-mail votes (see Figure A.30). This Process is small, but fairly complex and will provide examples for many of the features of BPMN. There are some unusual features of this business process, such as infinite loops. Although not a typical process, it will help illustrate that BPMN can handle simple and unusual business processes and still be easily understandable for readers of the Diagram. The sections below will isolate segments of the Process and highlight the modeling features as the workings of the Process is described. In addition, samples of BPEL4WS code are provided to demonstrate how a BPMN Diagram maps to BPEL4WS.
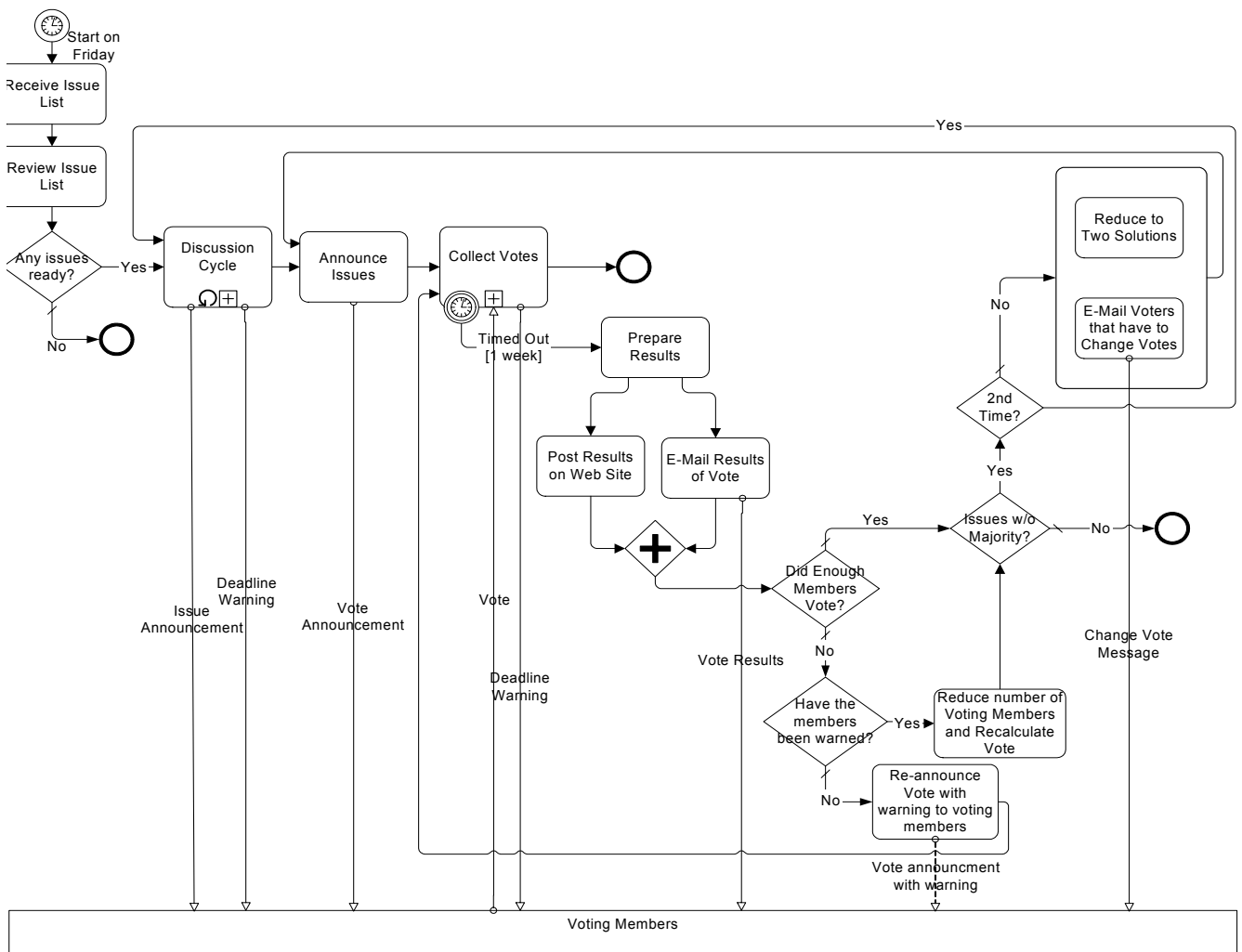
Figure A.30 - E-Mail Voting Process

The Process has a point of view that is from the perspective of the manager of the Issues List and the discussion around this list. From that point of view, the voting members of the working group are considered as external Participants who will be communicated with by messages (shown as Message Flow).

A.2.1   The Beginning of the Process

The Process starts with Timer Start Event that is set to trigger the Process every Friday (see Figure A.31).
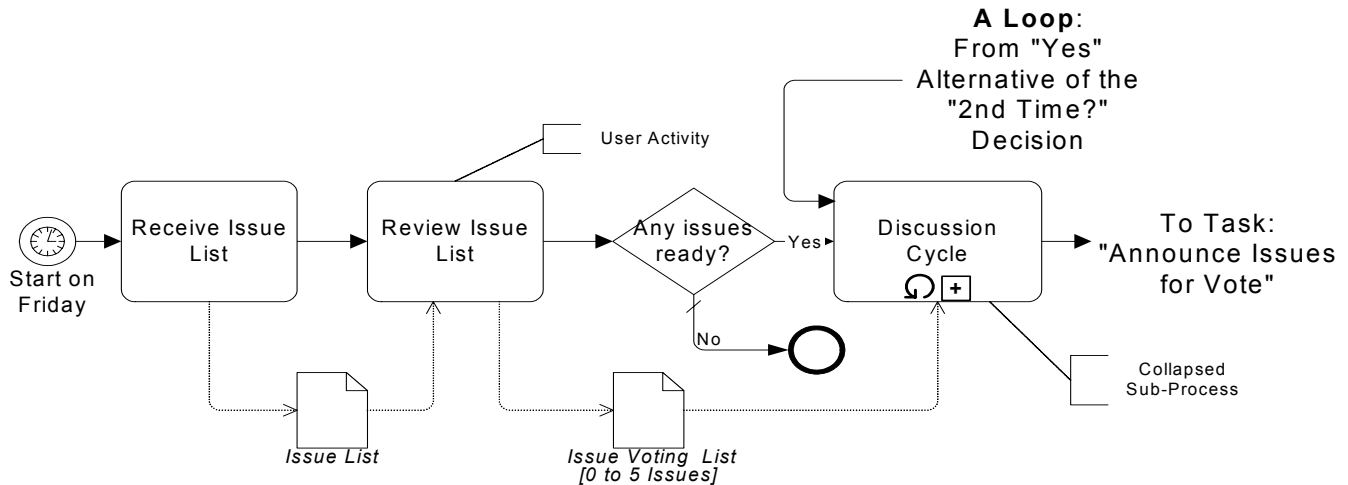
Figure A.31 - The Start of the Process

The Issue List Manager will review the list and determine if there are any issues that are ready for going through the discussion and voting cycle. Then a Decision must be made. If there are no issues ready, then the Process is over for that week--to be taken up again the following week. If there are issues ready, then the Process will continue with the discussion cycle. The "Discussion Cycle" Sub-Process is the first activity after the "Any issues ready?" Decision and this Sub-Process has two incoming Sequence Flow, one of which originates from a downstream Decision and is thus part of a loop. It is one of a set of five complex loops that exist in the Process. The contents of the "Discussion Cycle" Sub-Process and the activities that follow will be described below.

Mapping to BPEL4WS

BPEL4WS *processes* must begin with a *receive* activity for instantiation (i.e., it "bootstraps" itself). The "E-Mail Voting Process" is scheduled to start every Friday as shown by the Timer Start Event. Therefore, an additional Process will have to be created and implemented that will run indefinitely and will send a starting message with the list of Issues to the "E-Mail Voting Process" every Friday. Figure A.32 shows this Process as starting that the beginning of the Working Group and continuing until the end of the Working Group. Even this Process needs a message to be sent to it to signal the start of the Working Group. There may be another Process defined that sends that message, but that Process is not shown here. In addition, the mapping from the Starter Process to BPEL4WS is not shown here.
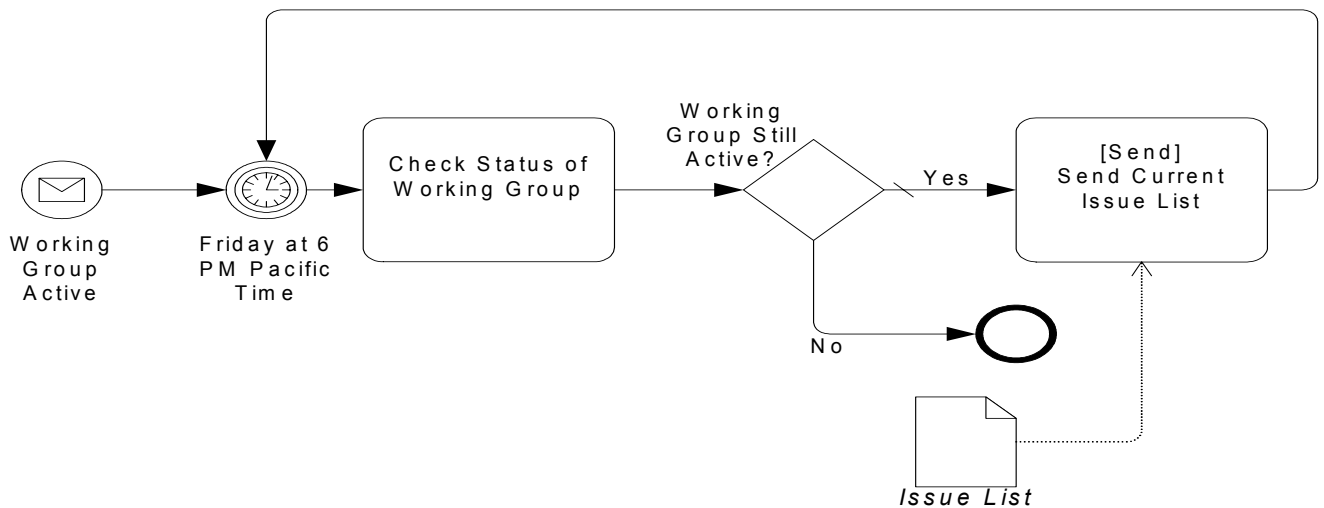
Figure A.32 - The Ongoing Starter Process

- Within the main Process (see Figure A.31), the "Receive Issue List" Task will map to a BPEL4WS *receive* that has its *createInstance* attribute set to "yes." This will receive starting message and start the *process*.

- This *receive* will be placed inside a *sequence* since other activities follow the activity. The *message* to be received will contain all the *variable parts* that will be used in the *process* and their initialized values.

Note – the names of BPD objects have all non-alphanumeric characters stripped from them when they are mapped to BPEL4WS *name* elements to match the BPEL4WS element restrictions.

The modeler-defined properties of the Process will be placed in a BPEL4WS *variables* element named "processData." The same *variables* element will be used in all derived *processes* in this example.

- The "Review Issue List" Task will map to a BPEL4WS *invoke*. This TaskType is User, which means that the *invoke* will be synchronous and an *outputVariable* included.

Mapping an Exclusive Gateway (Decision)

- The "Any Issues Ready?" Exclusive Gateway (Decision) will map to a BPEL4WS *switch*.

- The Gate for the "No" Sequence Flow will map to the *otherwise case* of the *switch*. This *otherwise* will only contain an *empty activity* since there is nothing to do and the Process is over.

Note that *empty* does not have any corresponding activity in the BPMN Diagram, but is derived through the Diagram configuration.

- The Gate for the "Yes" Sequence Flow will map to other *case* for the *switch*. This *case* will have a *condition* that checks the number of issues that are ready. This *case* will handle the remainder of the Process that is shown in Figure A.30.

This is done because the *switch* is a block structure and needs a definitive ending point and since the *otherwise* is connected to the end of the Process, then the end of the Process is the ending point that the *case* must use. The actual activities that make up the rest of the Process will be distributed among a set of BPEL4WS *processes* instead of all being

within the *case*. The *case* will only contain an *invoke* that will call another *process* (as a web service). The distribution of the Process activities is due to the overall Diagram configuration that includes three upstream Sequence Flow that define some interleaving loops.

The Impact of Interleaved Loops

If the loop shown in this section of the model were merely a simple loop, and perhaps the only loop, then a BPEL4WS *while* would be used to handle the loop. In this situation, though, the looping is handled through a set of derived *processes* that are accessed by *invoking* them (as a web service). There would no specific Diagram element to represent these derived *processes*; indeed, a modeler would not want to create a set of related Processes to handle complex looping. While an execution engine can easily handle a complex set of language documents and elements, a business person developing and monitoring this process will want to see the Process in an easy-to-read format (such as BPMN) that contains the information in a more comprehensive, less distributed format. See "Interleaved Loops" on page 197 for details about how interleaved loops are mapped to BPEL4WS.

In this example, all derived *processes* will be named "[(target of loop) activity.Name]_Derived_Process." Any naming scheme will work as long as all the *processes* have unique names.

- Thus, to handle the rest of the Process, a derived *nested process* named "Discussion_Cycle_Derived_Process" is created and then

- A BPEL4WS *invoke* is used to access this *process* from the "Yes" *case* of the "Any issues ready?" *switch*.

We shall see that later in the Process the same *process* is accessed through another *invoke*, marking the source of the loop.

All the sub-processes and derived processes in the BPEL4WS documents must be started with the *receive* of a message and then a *reply* to send a message back to the calling *process*.

- This means that a *receive* will be the first *activity* inside a *sequence* that will be the main *activity* of these *process*es. These *receive* activities will have the *createInstance* attribute set to "Yes." A named "internal," a portType name "processPort" will be created to support all of these process to process communications. The WSDL operations that will support these communications will all be named "call_<process name>" (as noted above, the processes are actually spawned).

The "Discussion Cycle" Sub-Process shown in Figure A.31 will continue the *sequence* (after the instantiating *receive*) for the "Discussion_Cycle_Derived_Process" *process*.

- Since "Discussion Cycle" is a Sub-Process it will map to a separate BPEL4WS *process* that is access through an *invoke*.

Mapping an Activity Loop Condition

The "Discussion Cycle" Process has a loop marker. In this situation, the looping mechanism is simple. The attributes of the Sub-Process will tell us the details. The "Discussion Cycle" Sub-Process's relevant attributes are: LoopType: "Standard"; LoopCondition: DiscussionOver = "FALSE"; and TestTime: "After."

- This means that the *invoke* that calls the *process* will be enclosed within a *while* activity when the BPEL4WS is derived.
    - The LoopType will map to a BPEL4WS *while*. The LoopCondition of the Process (as shown above) will map to the "DiscussionOver = False" will be the condition for the *while*.

The default value for the "DiscussionOver" property is False, thus an activity within the Sub-Process will have to change it to True before the *while* loop is over. The logical opposite of the expression that is shown in the Sub-Process attributes is used since the EvaluationCondition property is "after." However, a *while* will test the condition prior to running the activity within. This means that to insure that the activity is always performed at least once (to mimic the behavior of an "until") a LoopCounter variable will always be added to a the while condition for an BPMN activity that has its TestTime attribute set to "After."

- The LoopCounter will be initialized to zero, and an *assign* will be added to the *sequence* prior to the *while* element.

- The *activity* of the *while* will be changed to a *sequence*, with the *invoke* for the Sub-Process, which is

  • Followed by an *assign* that will increment the LoopCounter variable, inside the *sequence*.

We will look into the details of the "Discussion Cycle" Sub-Process in Section A.2.2, "The First Sub-Process," on page 215.

BPEL4WS Sample for the Beginning of the Process

Example A.9 displays some sample BPEL4WS code that reflects the portion of the Process that was just discussed and is shown in Figure A.31.

```
<process name="EMailVotingProcess">
  <!-- The Process data is defined first-->
  <sequence>
    <!--This starts the beginning of the Process. The process that sends the
        starting message every Friday is related to the Timer Start Event and is
        not shown here.-->
    <receive partnerLink="Internal" portType="tns:processPort"
          operation="receiveIssueList" variable="processData" createInstance="Yes"/>
    <invoke name="ReviewIssueList" partnerLink="Internal"
          portType="tns:internalPort" operation="sendIssueList"
          inputVariable="processData" outputVariable="processData"/>
    <switch name="Anyissuesready">
      <!-- name="Yes" -->
      <case condition="bpws:getVariableProperty(ProcessData,NumIssues)>0">
        <!--A chunk of this process is separated into a derived process so that it can be
        called from a complex loop. Thus, it is called from here and from "Collect Votes"
        as part of a loop-->
        <invoke name="Discussion_Cycle_Derived_Process" partnerLink="Internal"
              portType="tns:processPort"
              operation="call_Discussion_Cycle_Derived_Process" inputVariable="processData"
              outputVariable="processData"/>
      </case>
      <!--name="No" -->
      <otherwise>
        <!--This is one of the two ways to the end of the Process-->
        <empty/>
      </otherwise>
    </switch>
  </sequence>
</process>

<process name="Discussion_Cycle_Derived_Process">
  <!-- The Process data is defined first-->
  <sequence>
    <receive partnerLink="Internal" portType="tns:processPort"
          operation="call_Discussion_Cycle_Derived_Process" variable="processData"
          createInstance="Yes"/>
    <!--The first Sub-Process has a loop condition, so it is within a while-->
    <assign name="Discussion_Cycle_initialize_loopCounter">
      <copy>
        <from expression="0"/>
        <to variable="Discussion_Cycle_loopCounter" part="loopCounter" />
      </copy>
    </assign>
    <!--Since the TestTime is "After" the Sub-Process has to be performed before the
        while-->
    <invoke name="Discussion_Cycle" partnerLink="Internal"
```

```
            portType="tns:processPort operation="call_Discussion_Cycle"
            inputVariable="processData" outputVariable="processData"/>
  <while condition="bpws:getVariableProperty(ProcessData,DiscussionOver)=false">
     <!--This calls the first Sub-Process-->
     <sequence>
       <invoke name="Discussion_Cycle" partnerLink="Internal"
            portType="tns:processPort operation="call_Discussion_Cycle"
            inputVariable="processData" outputVariable="processData"/>
       <assign>
        <copy>
          <from expression=
               "bpws:getVariableProperty(Discussion_Cycle_loopCounter,LoopCounter)+1"/>
          <to variable="Discussion_Cycle_loopCounter" part="LoopCounter"/>
        </copy>
       </assign>
     </sequence>
  </while>
  <!--This calls the first another derived process to handle the rest of the
      work-->
  <invoke name="Announce_Issues_Derived_Process" partnerLink="Internal"
        portType="tns:processPort" operation="call_Announce_Issues_Derived_Process"
        inputVariable="processData" outputVariable="processData"/>
  <reply partnerLink="Internal" portType="tns:processPort"
        operation="call_Discussion_Cycle_Derived_Process" variable="processData"
        createInstance="Yes"/>
  </sequence>
</process>
<!--A lot of other activity follows (not shown)-->
```

Example A.9 - BPEL4WS Sample for Beginning of E-Mail Voting Process

A.2.2   The First Sub-Process

Figure A.33 shows the details of the "Discussion Cycle" as an Expanded Sub-Process.
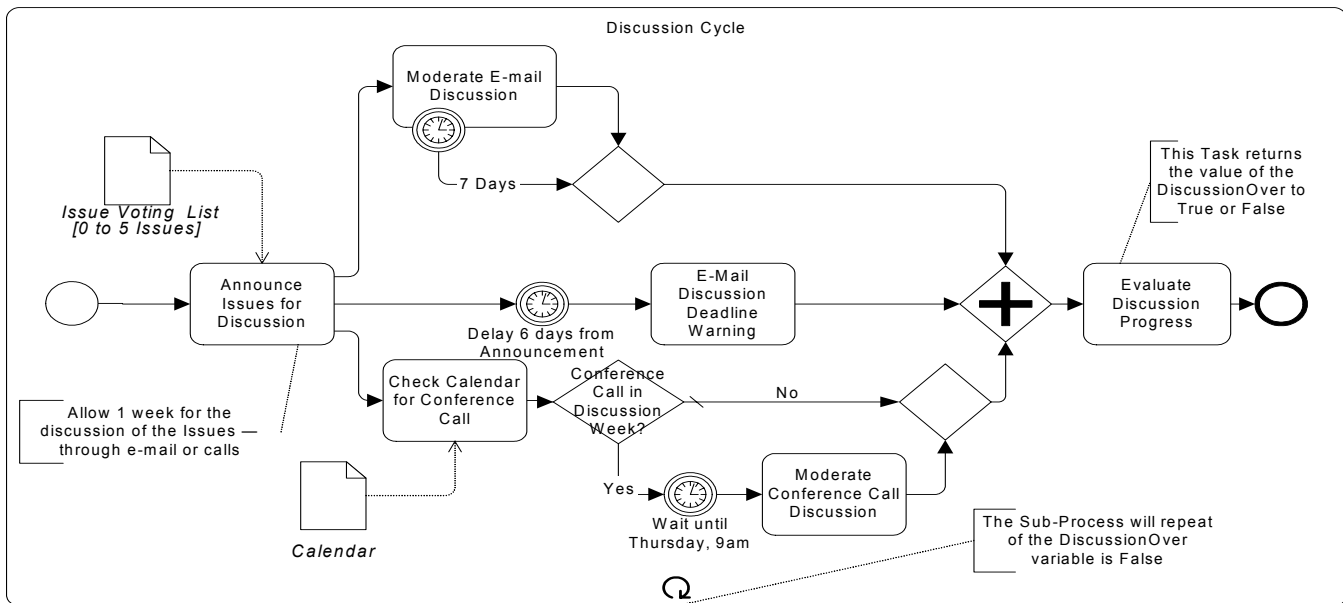
Figure A.33 - "Discussion Cycle" Sub-Process Details

The Sub-Process starts of with a Task for the Issue List Manager to send an e-mail to the working group that a set of Issues are now open for discussion through the working group's message board. Since this Task sends a message to an outside Participant (the working group members), an outgoing Message Flow is seen from the "Discussion Cycle" Sub-Process to the "Voting Members" Pool in Figure A.30. Basically, the working group will be discussing the issues for one week and proposing additional solutions to the issues. After the first Task, three separate parallel paths are followed, which are synchronized downstream. This is shown by the three outgoing Sequence Flow for that activity.

The top parallel path in the figure starts with a long-running Task, "Moderate E-mail Discussion," that has a Timer Intermediate Event attached to its boundary. Although the "Moderate E-Mail Discussion" Task will never actually be completed normally in this model, there must be an outgoing Sequence Flow for the Task since Start and End Events are being used within the Process. This Sequence Flow will merged with the Sequence Flow that comes from the Timer Intermediate Event. A merging Exclusive Gateway is used in this situation because the next object is a joining Parallel Gateway (the diamond with the cross in the center) that is used to synchronize the three parallel paths. If the merging Gateway was not used and both Sequence Flow connected to the joining Gateway, the Process would have been stuck at the joining Gateway that would wait for a Token to arrive from each of the incoming Sequence Flow.

The middle parallel path of the fork contains an Intermediate Event and a Task. A Timer Intermediate Event used in the middle of the Process flow (not attached to the boundary of an activity) will cause a delay. This delay is set to 6 days. The "E-Mail Discussion Deadline Warning" Task will follow. Again, since this Task sends a message to an outside Participant, an outgoing Message Flow is seen from the "Discussion Cycle" Sub-Process to the "Voting Members" Pool in Figure A.30.

The bottom parallel path of the fork contains more than one object, first of which is Task where the issue list manager checks the calendar to see if there is a conference call this week. The output of the Task will be an update to the variable "ConCall," which will be true or false. After the Task, an Exclusive Gateway with its two Gates follows. The Gate for labeled "default" Flow directly to an merging Exclusive Gateway, for the same reason as in the top parallel path. The Gate for the "Yes" Sequence Flow will have a *condition* that checks the value of the "ConCall" variable (set in the

previous Task) to see if there will be a conference call during the coming week. If so, the Timer Intermediate Event indicates delay, since all conference calls for the working group start at 9am PDT on Thursdays. The Task for moderating the conference call follows the delay, which is followed the merging Gateway.

The merging Gateways in the top and bottom paths and the "E-Mail Discussion Deadline Warning" Task all flow into a joining Gateway. This Gateway waits for all three paths to complete before the Process Flow to the next Task, "Evaluate Discussion Progress." The issue list manager will review the status of the issues and the discussions during the past week and decide if the discussions are over. The DiscussionOver variable will be set to TRUE or FALSE, depending on this evaluation. If the variable is set to FALSE, then the whole Sub-Process will be repeated, since it has looping set and the loop condition will test the DiscussionOver variable.

Mapping to BPEL4WS

- The "Discussion Cycle" Sub-Process itself maps to a BPEL4WS *process*.

  Because it is a Sub-Process within a higher-level Process (the "E-Mail Voting" Process), it is *invoked* from the higher-level Process. The *invoke* sends a message from one (higher-level) BPEL4WS *process* to the other (lower-level) *process* for instantiation.

- This means that the *process* being instantiated must have a *receive* to start it off.

- The *process* being instantiated must have a *reply* to end it, since it is being synchronously called.

  The *receive* and *reply* are not actually shown in the BPMN Diagram, but it is derived from this *invoke* relationship of "Discussion Cycle" Process being a Sub-Process to the "E-Mail Voting" Process.

- Given this, the *activity* of the BPEL4WS *process* will be a *sequence* with the derived *receive* as the first *activity*.

The Diagrams elements of Figure A.33 will determine the remaining activity(ies) of the sequence.

- The Sub-Process starts off with a Task, which maps to a BPEL4WS *invoke* (which is after the automatically generated receive that starts the *process*).

- After the first Task, three separate parallel paths are followed. The forking of the flow marks the start of a BPEL4WS *flow*. The *flow* will extend until the Parallel Gateway, which joins the three paths.

The Upper Parallel Path

In the upper parallel path of the fork, the Task, "Moderate E-mail Discussion," has a Timer Intermediate Event attached to its boundary. Because of this,

- the Task is placed in its own *scope* with a *faultHandlers*.

- The Task itself is mapped to a BPEL4WS *invoke* (synchronous), and will be placed in a lower-level *flow*, for reasons described below.

The Timer Intermediate Event must be set up to create a *fault* at the appropriate time. To do this,

- An *eventHandlers* is added to the *scope*.
  - An *onAlarm* is included in the *eventHandlers* and the *for* attribute is set to the duration that is defined in the Timer Intermediate Event.
    - The *onAlarm* contains a *throw* with a fault name after the Intermediate Event with "_Exit" appended.

The *catch* of a *faultHandlers* will be triggered by the *fault* generated by the above *throw*. Since the Timer Intermediate Event leads direction to the Exclusive Gateway, there is no specific activity that must be performed in response the to time-out. The main purpose is to exit the Task. Thus,

- A *faultHandlers* is added to the *scope*.
    - The *catch* in the *faultHandlers* has a *faultName* set to Intermediate Event with "_Exit" appended.
        - the *catch* will contain an *empty* activity.

The Middle Parallel Path

The middle parallel path of the fork has a string of two objects.

- Even though this series of objects appears in the middle of a BPEL4WS *flow*, they will be place within a *sequence* element.

In these situations, the *sequence* will continue until there is a location in the Diagram where there are multiple incoming Sequence Flow. When more than one Sequence Flow converge it marks the end of a BPEL4WS structure (as determined by structures that have been created by upstream objects). In this case, the Parallel Gateway also marks the end of the higher-level *flow*. The *sequence* will be listed in the higher-level *flow* without a *source* sub-element. This means that the *sequence* will be instantiated when the higher-level *flow* begins since it has no dependencies on any other *activity*. The *sequence* will have two activities:

- First, the Timer Intermediate Event used in this situation will map to a BPEL4WS *wait* (set to 6 days).

- Second, the "E-Mail Discussion Deadline Warning" Task will map to an *invoke* that follows the *wait*. In addition, this *invoke* can be asynchronous since a response is not required. This means that the *outputVariable* will not be included.

This middle path of the fork could have been configured in BPEL4WS without a *sequence* and with *links* instead. This is an example of a situation where a BPMN configuration may derive two possible BPEL4WS configurations. Since both BPEL4WS configurations will handle the appropriate behavior, it is up to the implementation of the BPMN to BPEL4WS derivation to determine which configuration will be used. BPMN does not provide any specific recommendation in these situations. However, the lower parallel path of the Process can also be modeled with a *sequence* or with *links*, and, to show how links would be used, this section of the Process will be mapped to elements in a *flow* that have dependencies specified by *links*.

The Lower Parallel Path

The lower parallel path of the fork has a number of objects and, as just described above, will be mapped to BPEL4WS elements connected with *links*. The path also contains a Decision, which can map to a *switch*, as will happen later in the process, but in this situation the Decision is mapped to *links* controlled by *transitionConditions*.

- The first object is a Task, which will map to an *invoke* (synchronous) that has two *source* elements referring to two of the *links*. There are two Target *links* because the Task is followed by the Gateway with its two Gates. This is done instead of a *switch* with a *case* and an *otherwise*.
    - The ConditionExpression for the Gate labeled "Yes" will map to the *source* element's *transitionCondition*. The expression checks the value of the "ConCall" property (set in the previous Task) to see if there will be a conference call during the coming week.
    - The Gate labeled "No" has a condition of default. For a *switch*, this would map to the *otherwise* element. However, since a *switch* is not being used, the *source* element's *transitionCondition* must be the inverse of all the other *transitionConditions* for the activity. The expression of the other *source* will be placed inside a "not" function.

The *invoke* will be listed in the higher-level *flow* without a *source* sub-element. This means that the *invoke* will be instantiated when the higher-level *flow* begins since it has no dependencies on any other *activity*. The remaining elements of the higher-level *flow* will have a *source* element. Thus, they will not be instantiated until the source of the *link* has completed.

- The "Yes" Gate from the Gateway leads to a Timer Intermediate Event, which will map to a *wait*.
  - The *for* element of the wait will set to for 9am PDT on the next Thursday.
  - This *wait* will have a *target* element that corresponds to the *target* element from the previous *invoke*.
  - The *wait* will also have a *target* element to link to the following *invoke*.
- The "No" Gate from the Gateway leads to a merging Exclusive Gateway, which means that nothing is expected to happen down this path. Thus, this will map to an *empty*.
  - This *empty* will have a *target* element that corresponds to the *target* element from the previous *invoke*.
- The Task for moderating the conference call follows the *wait*, which will map to an *invoke* (synchronous).
  - This *invoke* will have a *target* element that corresponds to the *target* element from the previous *wait*.

There are three link elements in the *flow*:

- One *link* will have a source of the first *invoke* and a target of the *wait*.
- One *link* will have a source of the first *invoke* and a target of the *empty*.
- One *link* will have a source of the first *wait* and a target of the last *invoke*.

As mentioned above, the Parallel Gateway marks the end of the *flow*.

Finally, there will be a *reply* at the end of the *sequence* that corresponds to the initial *receive* and lets the parent *process* know that the (sub) *process* has been completed.

After the Parallel Paths are Joined

The Task "Evaluate Discussion Progress" is intended to occur only when all the parallel paths have completed, and thus, it will

- Map to an *invoke* that follows the closing of the *flow*.

BPEL4WS Sample for the First Sub-Process

Example A.10 displays some sample BPEL4WS code that reflects the portion of the Process as described above and shown in Figure A.33.

```xml
<process name="Discussion_Cycle">
  <!-- The Process data is defined first-->
  <sequence>
    <receive partnerLink="Internal" portType="tns:processPort"
             operation="call_Discussion_Cycle" variable="processData" createInstance="Yes"/>
    <invoke name="AnnounceIssuesforDiscussion" partnerLink="WGVoter"
            portType="tns:emailPort" operation="sendDiscussionAnnouncement"
            inputVariable="processData"/>
    <flow>
      <links>
        <link name="CheckCalendarforConferenceCalltoWaituntilThursday,9am"/>
        <link name="CheckCalendarforConferenceCalltoEmpty"/>
        <link name="WaituntilThursday9amtoModerateConferenceCallDiscussion"/>
      </links>
      <!-- This is the first of the three paths of the fork. -->
      <scope>
        <invoke name="ModerateEmailDiscussion" partnerLink="internal"
                portType="tns:internalPort" operation="sendDiscussion"
                inputVariable="processData" outputVariable="processData"/>
        <faultHandlers>
          <catch faultName="7Days_Exit">
            <empty/>
          </catch>
        </faultHandlers>
        <eventHandlers>
          <onAlarm for="tns:OneWeek">
            <throw faultName="7Days_Exit"/>
          </catch>
        </eventHandlers>
      </scope>
      <!-- This is the second of the three paths of the fork. -->
```

```
    <sequence>
      <wait name="Delay6daysfromDiscussionAnnouncement" for="P6D"/>
      <invoke name="EMailDiscussionDeadlineWarning" partnerLink="WGVoter"
            portType="tns:emailPort" operation="sendDiscussionWarning"
            inputVariable="processData">
      </invoke>
    </sequence>
    <!-- This is the third of the three paths of the fork. -->
    <invoke name="CheckCalendarforConferenceCall" partnerLink="internal"
            portType="tns:internalPort" operation="receiveCallSchedule"
            inputVariable="processData" outputVariable="processData">
      <source linkName="CheckCalendarforConferenceCalltoWaituntilThursday9am"
          transitionCondition="bpws:getVariableProperty(processData,conCall)=true"/>
      <source linkName="CheckCalendarforConferenceCalltoEmpty"
        transitionCondition="not(bpws:getVariableProperty(processData,conCall)=true)"/>
    </invoke>
    <!-- name="Yes" -->
    <wait name="WaituntilThursday9am" for="P6DT9H">
      <target linkName="CheckCalendarforConferenceCalltoWaituntilThursday9am">
      <source linkName="WaituntilThursday9amtoModerateConferenceCallDiscussion"/>
    </wait>
    <invoke name="ModerateConferenceCallDiscussion" partnerLink="internal"
            portType="tns:internalPort" operation="sendConCall"
            inputVariable="processData" outputVariable="processData">
      <target linkName="WaituntilThursday9amtoModerateConferenceCallDiscussion"/>
    </invoke>
    <!-- name="otherwise" -->
    <empty>
      <target linkName="CheckCalendarforConferenceCalltoEmpty"/>
    </empty>
  </flow>
  <invoke name="EvaluateDiscussionProgress" partnerLink="internal"
        portType="tns:internalPort" operation="receiveDiscussionStatus"
        inputVariable="processData" outputVariable="processData"/>
  <reply partnerLink="Internal" portType="tns:processPort"
        operation="call_Discussion_Cycle" variable="processData"/>
  </sequence>
</process>
```

Example A.10 - BPEL4WS Sample of "Discussion Cycle" Sub-Process Details

A.2.3   The Second Sub-Process

Figure A.34 shows the next section of the Process, which includes the expanded details of the "Collect Votes" Sub-Process.

A Loop:
From Unnamed
Sub-Process
(parallel box)

Collect Votes

Check Calendar
for Conference
Call

Conference
Call in Voting
Week?

No

Calendar

Yes

Wait until
Thursday, 9am

Moderate
Conference Call
Discussion

Announce
Issues

From Sub-Process:
"Discussion Cycle"

Moderate E-mail
Discussion

Delay 6 Days

E-Mail Vote
Deadline Warning

Receive Vote

Increment Tally

Vote

Vote Tally

Timed Out
[1 week]

Prepare
Results

A Loop:
From Task:
"Re-announce Vote with
warning to voting members"

E-Mail Results
of Vote

Post Results
on Web Site

To Decision:
"Did Enough
Members Vote?"

Figure A.34 - "Collect Votes" Sub-Process Details

This part of the process starts out with a Task for the issue list manager to send out an e-mail to announce to the working group, and the voting members in particular, which lets them know that the issues are now ready for voting. Since this Task sends a message to an outside Participant (the working group members), an outgoing Message Flow is seen from the "Announce Issues" Task to the "Voting Members" Pool in Figure A.30. This Task is also a target for one of the complex loops in the Process.

The "Collect Votes" Sub-Process follows the Task, and is also a target of one of the looping Sequence Flow. This Sub-Process is basically a set of four parallel paths that extend from the beginning to the end of the Sub-Process.

The first branch of the fork leads to a Decision that determines whether or not a conference call will occur during the upcoming week, after the Working Group's schedule has been checked. Basically, if there was a call last week, then there will not be a call this week and vice versa. The appropriate variable that was updated in the "Discussion Cycle" Process will be used again.

The second and third branches forks work the same way as the similar activities in the "Discussion Cycle" Sub-Process, except that the "Moderate E-Mail Discussion" Task does not have a Timer Intermediate Event attached. This is not necessary since the whole Sub-Process is interrupted after 7 days through the Intermediate Event attached to the Sub-Process boundary. The "E-Mail Vote Deadline Warning" Task sends a message to an outside Participant (the working group members), thus, an outgoing Message Flow is seen from the "Collect Votes" Sub-Process to the "Voting Members" Pool in Figure A.30.

The fourth branch of the fork is rather unique in that the Diagram uses a loop that does not utilize a Decision. Thus, it is, as it is intended to be, an infinite loop. The policy of the working group is that voting members can vote more than once on an issue; that is, they can change their mind as many times as they want throughout the entire week. The first Task in the loop receives a message from the outside Participant (the working group members), thus, an incoming Message Flow is seen from the "Voting Members" Pool to the "Collect Votes" Sub-Process in Figure A.30. The Timer Intermediate Event attached to the boundary of the Sub-Process is the mechanism that will end the infinite loop, since all work inside the Sub-Process will be ended when the time-out is triggered. All the remaining work of the Process is conducted after the time-out and Flow from the Timer Intermediate Event.

Figure A.34 shows that there are Two Tasks that follow the time-out. First, a Task will prepare all the voting results, then a Task will send the results to the voting members. A Document Object, "Issue Votes," is shown in the Diagram to illustrate how one might be used, but it will not map to anything in the execution languages. The remaining activities of the Process will be described in the next section.

Mapping to BPEL4WS

The Loops Cause Derived Sub-Processes

- The first Task of this section of the Process is also a target for one of the complex loops in the Process, thus, it will map to an *invoke* (asynchronous) that is placed inside another derived *process* ("Announce_Issues_Derived_Process").

- This derived *process* will be *invoked* from "Discussion_Cycle_Derived_Process," after the "Discussion Cycle" process has been completed, as part of the Normal Flow and then from another part of the Process as part of the looping flow.

  - Thus, "Announce_Issues_Derived_Process" will require a (instantiation) *receive* to accept the message from "Discussion_Cycle_Derived_Process" and from "Issues_wo_Majority_Derived_Process" (as we shall see later).

- The "Collect Votes" Sub-Process follows the Task, but is also a target of one of the looping Sequence Flow. Thus, it will also be set inside a derived *process* ("Collect_Votes_Derived_Process").

  - In addition, "Collect_Votes_Derived_Process" will require a (instantiation) *receive* to accept the message from "Announce_Issues_Derived_Process" and from the fault handler of "Collect Votes" (as we shall see later).

- The "Collect Votes" Sub-Process will map to an *invoke* (asynchronous) and the details will be in a *process* referenced through the *invoke*.

The BPEL4WS Sample of the Derived Sub-Processes

Example A.11 shows sample BPEL4WS code that defines the two derived *processes*.

```xml
<process name="Announce_Issues_Derived_Process">
  <!-- This starts the middle section of the Process and is call from
       the first time and then from "Collect Votes" during a loop-->
  <!-- The Process data is defined first-->
    <sequence>
      <receive partnerLink="Internal" portType="tns:processPort"
            operation="call_Announce_Issues_Derived_Process"
            variable="processData" createInstance="Yes"/>
      <invoke name="AnnounceIssuesforVote" partnerLink="WGVoter" portType="tns:emailPort"
             operation="sendVoteAnnouncement" inputVariable="processData"/>
      <invoke name="Collect_Votes_Derived_Process" partnerLink="Internal"
             portType="tns:processPort"
             operation="call_Collect_Votes_Derived_Process" inputVariable="processData"/>
      <reply partnerLink="Internal" portType="tns:processPort"
            operation="call_Announce_Issues_Derived_Process"
            variable="processData" createInstance="Yes"/>
    </sequence>
</process>

<process name="Collect_Votes_Derived_Process">
  <!-- this calls the second Sub-Process and then continues. It is also
       called from "Collect Votes" as part of a loop-->
  <!-- The Process data is defined first-->
    <sequence>
      <receive partnerLink="Internal" portType="tns:processPort"
            operation="call_Collect_Votes_Derived_Process" variable="processData"
            createInstance="Yes"/>
      <invoke name="Collect_Votes" partnerLink="Internal" portType="tns:processPort"
             operation="call_Collect_Votes" inputVariable="processData"/>
      <reply partnerLink="Internal" portType="tns:processPort"
            operation="call_Collect_Votes_Derived_Process" variable="processData"
            createInstance="Yes"/>
    </sequence>
</process>
```

Example A.11 - BPEL4WS Sample that sets up the Access for the Second Sub-Process

The Paths of the Sub-Process

The "Collect Votes Sub-Process is basically a set of four parallel paths that extend from the beginning to the end of the Sub-Process.

- Thus, the *activity* for the *process* will be a *flow.*

The Upper Parallel Path

The first branch of this Sub-Process is basically the same as the upper parallel of the previous Sub-Process. An *invoke*, a *wait*, and an *empty* will be created. In addition, three *links* will be created to handle the dependencies between the elements, including the branching created by the Exclusive Gateway. See "The Lower Parallel Path" on page 218 for the details of the mappings.

The Middle Two Parallel Paths

The second and third branches of the fork are rather straightforward mappings of:

- Two Tasks to *invokes* (one synchronous and one asynchronous), and

- A Timer Intermediate Event to a *delay*.

- In addition, one *link* is created so that one of the *invokes* will wait for the *delay*.

The Lower Parallel Path

The fourth branch of the fork is the location the infinite loop.

- This loop will map to a BPEL4WS *while* with a *condition* of "1=0," which will always be false.

- Inside the *while* is a *sequence* of two *invokes* (one synchronous and one asynchronous), which are mapped from the two Tasks in the loop.

Exiting the Second Sub-Process

To exit out of the infinite loop and the whole "Collect Votes" Sub-Process,

- A *scope* will be wrapped around the main *flow* of the *process*, which will include an *eventHandlers* and a *faultHandlers*.

The Timer Intermediate Event must be set up to create a *fault* at the appropriate time. To do this,

- An *onAlarm* will be placed inside the *eventHandlers*. The timing of the *onAlarm* will be determined by the time setting in the Intermediate Event.

  - Within the *onAlarm*, a *throw* will a fault name after the Intermediate Event with "_Exit" appended.

- The *catch* element of the *faultHandlers* will be triggered by the *fault* generated by the above *throw*.

  - The *activity* for the *catch* will be a *sequence* and will be the source of all the remaining activities of the Process, since all the remaining Sequence Flow begins from the Timer Intermediate Event.

    - The first three Tasks, as shown in the figure, will map to *invokes*. The latter two will be placed within a *flow*.

The Document Objects shown in the figure is not mapped into BPEL4WS. The remainder of the Process will be described in the next section.

BPEL4WS Sample for the Second Sub-Process

Example A.12 shows sample BPEL4WS code that defines the "Collect Votes" Sub-Process.

```xml
<process name="Collect_Votes">
  <!--This is a nested process for the E-Mail Voting collection. It consists of
      an all and a faultHandlers (for a time-out). The all will never complete
      normally since there is an infinite loop inside. The timeout is intended to
      be the normal way of ending the process-->
  <sequence>
    <receive partnerLink="Internal" portType="tns:processPort"
             operation="call_Collect_Votes" variable="processData" createInstance="Yes"/>
    <scope>
      <flow>
        <links>
          <link name="Delay6daysfromVoteAnnouncementtoEMailVoteDeadlineWarning"/>
          <link name="CheckCalendarforConferenceCalltoWaituntilThursday9am"/>
          <link name="CheckCalendarforConferenceCalltoEmpty"/>
          <link name="WaituntilThursday9amtoModerateConferenceCallDiscussion"/>
        </links>
        <!--This is the first of the four paths of the fork. -->
        <invoke name="CheckCalendarforConferenceCall" partnerLink="internal"
                portType="tns:internalPort" operation="receiveCallSchedule"
                inputVariable="processData" outputVariable="processData">
          <source linkName="CheckCalendarforConferenceCalltoWaituntilThursday9am"
             transitionCondition="bpws:getVariableProperty(processData,conCall)=true"/>
          <source linkName="CheckCalendarforConferenceCalltoEmpty"
            transitionCondition="not(bpws:getVariableProperty(processData,conCall)=true)"/>
        </invoke>
        <!-- name="Yes" -->
        <wait name="WaituntilThursday9am" for="P6DT9H">
          <target linkName="CheckCalendarforConferenceCalltoWaituntilThursday9am">
          <source linkName="WaituntilThursday9amtoModerateConferenceCallDiscussion"/>
        </wait>
        <invoke name="ModerateConferenceCallDiscussion" partnerLink="internal"
                portType="tns:internalPort" operation="sendConCall"
                inputVariable="processData" outputVariable="processData">
          <target linkName="WaituntilThursday9amtoModerateConferenceCallDiscussion"/>
        </invoke>
        <!-- name="otherwise" -->
        <empty>
          <target linkName="CheckCalendarforConferenceCalltoEmpty"/>
        </empty>
        <!-- This is the second of the four paths of the fork. -->
        <invoke name="ModerateEMailDiscussion" partnerLink="internal"
                portType="tns:internalPort" operation="sendDiscussion"
                inputVariable="processData" outputVariable="processData"/>
        <!--This is the third of the four paths of the fork.-->
        <wait name="Delay6daysfromVoteAnnouncement" for="P6D">
          <source linkName="Delay6daysfromVoteAnnouncementtoEMailVoteDeadlineWarning"/>
        </wait>
```

```xml
      <invoke name="EMailVoteDeadlineWarning" partnerLink="WGVoter"
            portType="tns:emailPort" operation="sendVoteWarning"
            inputVariable="processData">
        <target linkName="Delay6daysfromVoteAnnouncementtoEMailVote DeadlineWarning"/>
      </invoke>
      <!--This is the fourth of the four paths of the fork. This branch of the
            all is intended to be an infinite loop that is eventually
            interrupted by the Time Out. This is necessary since any voter can
            change their vote until the deadline. -->
      <while condition="1=0">
        <sequence>
          <receive name="ReceiveVote" partnerLink="WGVoter" portType="tns:emailPort"
                operation="receiveVote" variable="processData"/>
          <invoke name="IncrementTally" partnerLink="internal"
                portType="tns:internalPort" operation="sendReceiveTotal"
                inputVariable="processData" outputVariable="processData"/>
        </sequence>
      </while>
    </flow>
    <eventHandlers>
      <onAlarm for="P7D">
        <throw faultName="7days_Exit"/>
      </onAlarm>
    </eventHandlers>
    <faultHandlers>
      <catch faultName="7days_Exit">
        <!-- The BPMN Diagram shows that the Timer Intermediate Event connects directly
            to the rest of the Process. Thus, they will show up in this activity set. -->
        <sequence>
          <invoke name="PrepareResults" partnerLink="internal"
                portType="tns:internalPort" operation="sendReceiveResults"
                inputVariable="processData" outputVariable="processData"/>
          <flow>
            <invoke name="PostResultsonWebSite" partnerLink="internal"
                  portType="tns:internalPort" operation="postVotingResults"
                  inputVariable="processData"/>
            <invoke name="EMailResultsofVote" partnerLink="WGVoter"
                  portType="tns:emailPort" operation="sendVotingResults"
                  inputVariable="processData"/>
          </flow>

    <!--the rest of the process is not shown-->

    </faultHandlers>
  </scope>
  <reply partnerLink="Internal" portType="tns:processPort"
        operation="call_Collect_Votes" variable="processData" createInstance="Yes"/>
  </sequence>
</process>
```

Example A.12 - BPEL4WS Sample of the Second Sub-Process

A.2.4   The End of the Process

Figure A.35 shows the last section of the Process, which includes a complex set of Decisions and loops.
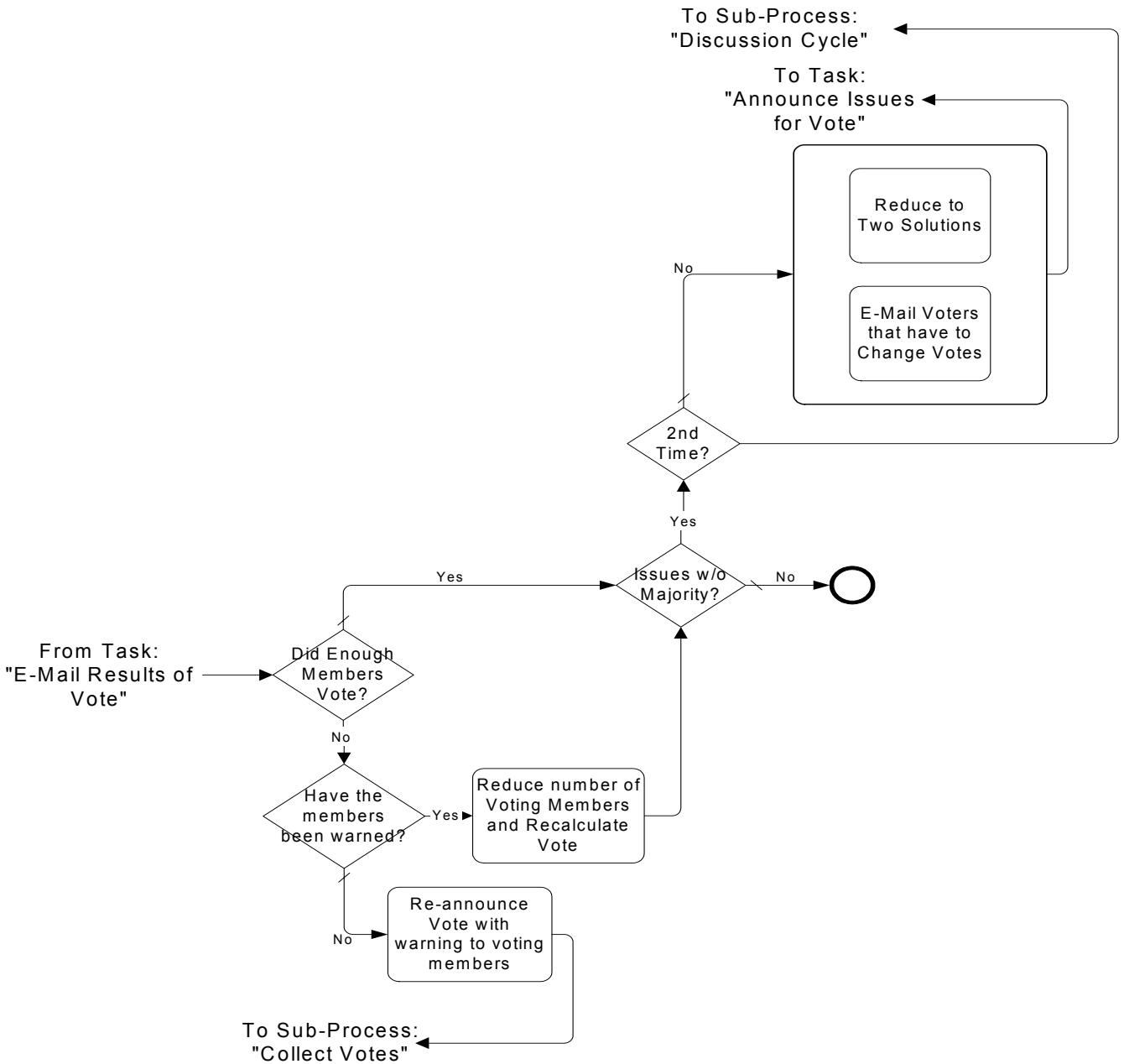
Figure A.35 - The last segment of the E-Mail Voting Process

This segment of the Process continues from where the last segment left off (as described in the section above). It contains four Decisions that interact with each other and create loops to upstream activities.

The first Decision, "Did Enough Members Vote?," is necessary since two-thirds of the voting members are required to approve any solution to an issue. If less than two-thirds of the voting members cast votes, which sometimes happens, the issues can't be resolved. This Decision Flow to another Decision for both of its Alternatives. The "No" Alternative is followed by the "Have the Members been Warned?" Decision. If a voting member misses a vote, they are warned. If they miss a second vote, they lose their status as a voting member and the voting percentages are recalculate through a Task ("Reduce number of Voting Members and Recalculate Vote"). If they haven't yet been warned, then a warning is sent and the voting week is repeated.

If all issues are resolved, then the Process is done. If not, then another Decision is required. The voting is given two chances before it goes back to another cycle of discussion. The first time will see a reduction of the number of solutions to the two most popular based on the vote (more if there are ties). Some voting members will have to change their votes just because their solution is no longer valid. These two activities are placed in a Sub-Process to show how a Sub-Process without Start and End Events can be used to create a simple set of parallel activities. Informally, this is called a "parallel box." It is not a special object, but another use of Sub-Processes. For simple situations, it can be used to show a set of parallel activities without the extra clutter of a lot of Sequence Flow. In actuality, these two Tasks cannot actually be done in parallel, but they are modeled this way to highlight the optional use of Start and End Events.

After the parallel box, the flow loops back to the "Collect Votes" Sub-Process. If there already has been two cycles of voting, then the process Flow back to the "Decision Cycle" Sub-Process.

Mapping to BPEL4WS

As mentioned above, the entire contents of this segment follow a Timer Intermediate Event, which means they are contained in the *faultHandlers* of the *scope* within the "Collect Votes" *process*.

- Each of the Decisions in this section will map to a BPEL4WS *switch*.

The First Decision

The first Decision, "Did Enough Members Vote?," Flow to another Decision for both of its Alternatives.

- Thus, each of the *switch cases* will contain another *switch*.

The "No" Alternative is followed by the "Have the Members been Warned?" Decision.

- Each Alternative from this Decision is followed by a Task, which maps to *Invokes* (one synchronous and the other asynchronous).

The "No (default)" Alternative leads to a loop.

- This looping is handled by using an *invoke* (asynchronous) to the "Collect_Votes_Derived_Process" *process*, which was created just for the purpose of this loop (since it is in the context of a more complex looping situation).

Notice that the "Issues w/o Majority?" Decision can be reached through the alternative paths from two different Decisions. This creates a situation that has two impacts on the Mapping to Execution Languages. First, it creates a section of the Process in which the Alternatives from two Decisions overlap. This is possible in a graph-structured Diagram like BPMN, but in a block-structured (and acyclic) language like BPEL4WS, these two sections cannot overlap because they have different block boundaries. This means that this section must be repeated in some way in both of the appropriate *switch case activities*. All these elements could be actually duplicated or they can be separated into a derived *process* and then *invoked* from the appropriate place. The later method was used in this example (see Example A.13 and Example A.14).

Note – At this point, BPMN does not specify whether a reused section of a BPMN Diagram should map to a derived *process*

that is *invoked* from each location of duplication, or whether the section should remain intact and be duplicated in each appropriate location. This is left up to the specific implementation of BPMN since both solutions will behave equivalently.

The second impact of the multiple incoming Sequence Flow into the "Issues w/o Majority?" Decision has to do with how the three visible loops are created (actually there are five loops). Normally, Sequence Flow loops will map to a BPEL4WS *while*. If there are multiple loops in the Process they have to be physically separated or completely nested because of the block-structured nature of the BPEL4WS looping elements. The alternative paths of the Decisions cannot be mixed and still maintain the BPEL4WS blocks they way that the end of the "E-mail Voting" Process mixes the paths.

A different type of looping mechanism is required. This method requires the creation of a set of derived *processes* that can reference each other and also themselves. In this way, a block-structured language can simulate a set of interleaving loops (as seen in a graph-structured Diagram).

- Thus, in this BPMN example, derived *processes* were created to mark places where loops can be targeted within the BPEL4WS code from the "downstream" elements.

- A BPEL4WS *invoke* is used to re-perform activities that had already been executed in the process.

BPEL4WS Sample for the End of the Process

Example A.13 displays the BPEL4WS code for first part of the end of the "E-Mail Voting Process."

```
    <!--This segment of the code is within the context of the "Collect
        Votes" nested process-->
<catch property="tns:OneWeek" type="duration">
    <!--The BPMN Diagram shows that the Timer Intermediate Event connects directly to the
rest of the Process. Thus, they will show up in this activity set-->
    <!--The first two actions are not shown-->
  <sequence>
    <invoke name="PrepareResults" partnerLink="internal" portType="tns:internalPort"
            operation="sendReceiveResults" inputVariable="processData"
            outputVariable="processData"/>
    <invoke name="EMailResultsofVote" partnerLink="WGVoter" portType="tns:emailPort"
             operation="sendVotingResults" inputVariable="processData"/>
    <switch name="DidEnoughMembersVote">
      <!-- name="No" -->
      <case condition="bpws:getVariableProperty(ProcessData,NumVoted)>
                      (.7)*(bpws:getVariableProperty(ProcessData,NumVWGM))">
```

```xml
        <switch name="Havethemembersbeenwarned">
          <!-- name="Yes" -->
          <case condition="bpws:getVariableProperty(ProcessData,VotersWarned)=true">
            <sequence>
              <invoke name="ReducenumberofVotingMembersandRecalculateVote"
                      partnerLink="internal" portType="tns:internalPort"
                      operation="sendReceiveNumVoters" inputVariable="processData"
                      outputVariable="processData"/>
              <!--Some elements of the process were separated into a derived
                  process since they would have been repeated. They would have
                  been repeated because they are arrived by alternative paths that
                  do not close a set of alternative paths. -->
              <invoke name="Issues_wo_Majority_Derived_Process" partnerLink="Internal"
                      portType="tns:processPort"
                      operation="call_Issues_wo_Majority_Derived_Process"
                      inputVariable="processData" outputVariable="processData"/>
            </sequence>
          </case>
          <!-- name="No (otherwise)" -->
          <otherwise>
            <sequence>
              <invoke name="ReannounceVotewithwarningtovotingmembers"
                      partnerLink="WGVoter" portType="tns:emailPort"
                      operation="sendReannounceVote" inputVariable="processData"
                      outputVariable="processData"/>
              <invoke name="Collect_Votes_Derived_Process" partnerLink="Internal"
                      portType="tns:processPort"
                      operation="call_Collect_Votes_Derived_Process"
                      inputVariable="processData" outputVariable="processData"/>
            </sequence>
          </otherwise>
        </switch>
      </case>
      <!-- name="Yes (otherwise)" -->
      <otherwise>
        <!-- Some elements of the process were separated into a derived process since they
             would have been repeated. They would have been repeated because they are
             arrived by alternative paths that do not close a set of alternative paths. -->
        <invoke process="Issues_wo_Majority_Derived_Process" partnerLink="Internal"
                portType="tns:processPort"
                operation="call_Issues_wo_Majority_Derived_Process"
                inputVariable="processData" outputVariable="processData"/>
      </otherwise>
    </switch>
  </sequence>
</catch>
```

Example A.13 - Sample BPEL4WS code for the last section of the Process

Example A.14 shows the BPEL4WS code for the Process from the "Issues w/o Majority?" Decision until the end of the Process or loops.

- The mappings are a fairly straightforward set of *switches*.

If all issues are resolved, then the Process is done. If not, then another Decision is required.

- The "parallel box," as is any forking situation, will map to a BPEL4WS *flow*.

After the parallel box, the flow loops back to the "Collect Votes" Sub-Process.

- This looping is handled by using an *invoke* (asynchronous) to the "Announce_Issues_Derived_Process" *process*, which was created just for the purpose of this loop.

If there has already been two cycles of voting, then the process Flow back to the "Decision Cycle" Sub-Process.

- This looping is handled by using an *invoke* (asynchronous) to the "Discussion_Cycle_Derived_Process" *process*, which was created just for the purpose of this loop.

Example A.13 displays the BPEL4WS code for the final derived *process* of the "E-Mail Voting Process."

```xml
<process name="Issues_wo_Majority_Derived_Process">
  <sequence>
    <receive partnerLink="Internal" portType="tns:processPort"
             operation="call_Issues_wo_Majority_Derived_Process" variable="processData"
             createInstance="Yes"/>
    <switch name="IssueswoMajority">
      <case name="Yes" condition="NoMajority=true">
        <switch name="2ndTime">
          <!-- name="Yes" -->
          <case condition="bpws:getVariableProperty(ProcessData,VotedOnce)=true">
            <!--This is done to do the complex looping situation. -->
            <invoke name="Discussion_Cycle_Derived_Process" partnerLink="Internal"
                    portType="tns:processPort"
                    operation="call_Discussion_Cycle_Derived_Process"
                    inputVariable="processData" outputVariable="processData"/>
          </case>
          <!-- name="No (otherwise)"-->
          <otherwise>
            <sequence>
              <flow>
                <invoke name="ReducetoTwoSolutions" partnerLink="internal"
                        portType="tns:internalPort" operation="sendReceiveSolutions"
                        inputVariable="processData" outputVariable="processData"/>
                <invoke name="EMailVotersthathavetoChangeVotes" partnerLink="WGVoter"
                        portType="tns:emailPort" operation="sendVoteWarning"
                        inputVariable="processData"/>
              </flow>
              <invoke process="Announce_Issues_Derived_Process" partnerLink="Internal"
                      portType="tns:processPort"
                      operation="call_Announce_Issues_Derived_Process"
                      inputVariable="processData" outputVariable="processData"/>
            </sequence>
          </otherwise>
        </switch>
      </case>
      <otherwise name="Nootherwise">
        <!-- This is one of the two ways to the end of the Process. -->
        <empty/>
      </otherwise>
    </switch>
    <reply partnerLink="Internal" portType="tns:processPort"
           operation="call_Issues_wo_Majority_Derived_Process" variable="processData"
           createInstance="Yes"/>
  </sequence>
</process>
```

Example A.14 - Sample BPEL4WS code for derived *process* for repeated elements

A.3 BPEL4WS for the E-Mail Voting Process

This section provides the complete BPEL4WS code for the example BPMN business process that is described in the "BPMN by Example" chapter.

```
<definitions
    targetNamespace="http://www.website.com"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <message name="processDataMessage">
    <part name="NumIssues" type="xsd:integer"/>
    <part name="NoMajority" type="xsd:boolean"/>
    <part name="VotedOnce" type="xsd:boolean"/>
    <part name="NumVoted" type="xsd:integer"/>
    <part name="VotersWarned" type="xsd:boolean"/>
    <part name="LoopCounter" type="xsd:integer"/>
  </message>
    <!--processDataMessage will be received with the following parts:
        NoMajority (set to false)
        VotedOnce (set to false)
        NumVoted (set to false)
        VotersWarned (set to false)
        LoopCounter (set to 0)
        starting message every Friday is not shown here.-->
</definitions>

<!-- The Main Process -->
<process name="EMailVotingProcess">
  <variables>
    <variable name="processData" messageType="processDataMessage"/>
    <!--processDataMessage will be received with the following parts:
        NumIssues (set to the number of unresolved Issues)
        NoMajority (set to false)
        VotedOnce (set to false)
        NumVoted (set to false)
        VotersWarned (set to false)
        LoopCounter (set to 0)
        starting message every Friday is not shown here.-->
  </variables>
  <sequence>
    <!--This starts the beginning of the Process. The process that sends the
        starting message every Friday is not shown here.-->
    <receive partnerLink="Internal" portType="tns:processPort"
          operation="receiveIssueList" variable="processData" createInstance="Yes"/>
    <invoke name="ReviewIssueList" partnerLink="Internal" portType="tns:internalPort"
          operation="sendIssueList" inputVariable="processData"
          outputVariable="processData"/>
    <switch name="AnyIssuesReady">
```

```xml
      <!--name="Yes" -->
      <case condition="bpws:getVariableProperty(ProcessData,NumIssues)>0">
        <!-- A chunk of this process is separated into a derived process so that
             it can be called from a complex loop. -->
        <invoke name="Discussion_Cycle_Derived_Process" partnerLink="Internal"
               portType="tns:processPort" operation="call_Discussion_Cycle_Derived_Process"
                inputVariable="processData" outputVariable="processData"/>
      </case>
      <!--name="No" -->
      <otherwise>
        <!--This is one of the two ways to the end of the Process.-->
        <empty/>
      </otherwise>
    </switch>
  </sequence>

<!-- A Derived Process -->
<process name="Discussion_Cycle_Derived_Process">
  <variables>
    <variable name="processData" messageType="processDataMessage"/>
    <variable name="Discussion_Cycle_loopCounter" messageType="loopCounterMessage"/>
  </variables>
  <sequence>
    <receive partnerLink="Internal" portType="tns:processPort"
            operation="call_Discussion_Cycle_Derived_Process" variable="processData"
             createInstance="Yes"/>
    <!--The first Sub-Process has a loop condition, so it is within a while-->
    <assign name="Discussion_Cycle_initialize_loopCounter">
      <copy>
        <from expression="0"/>
        <to variable="Discussion_Cycle_loopCounter" part="loopCounter" />
      </copy>
    </assign>
    <!--Since the TestTime is "After" the Sub-Process has to be performed before the
        while-->
    <invoke name="Discussion_Cycle" partnerLink="Internal"
          portType="tns:processPort operation="call_Discussion_Cycle"
          inputVariable="processData" outputVariable="processData"/>
    <while condition="bpws:getVariableProperty(ProcessData,DiscussionOver)=false">
      <!--This calls the first Sub-Process-->
      <sequence>
        <invoke process="Discussion_Cycle" partnerLink="Internal"
               portType="tns:processPort operation="call_Discussion_Cycle"
               inputVariable="processData" outputVariable="processData"/>
        <assign>
          <copy>
            <from expression=
                 "bpws:getVariableProperty(Discussion_Cycle_loopCounter,LoopCounter)+1"/>
```

```xml
            <to variable="Discussion_Cycle_loopCounter" part="LoopCounter"/>
          </copy>
        </assign>
      </sequence>
    </while>
    <!--This calls the first another derived process to handle the rest of the
        work-->
    <invoke name="Announce_Issues_Derived_Process" partnerLink="Internal"
            portType="tns:processPort" operation="call_Announce_Issues_Derived_Process"
            inputVariable="processData"  outputVariable="processData"/>
  </sequence>
</process>
</process>
<!-- A Derived Process -->
<process name="Announce_Issues_Derived_Process">
  <!-- This starts the middle section of the process. -->
  <variables>
    <variable name="processData" messageType="processDataMessage"/>
  </variables>
  <sequence>
    <receive partnerLink="Internal" portType="tns:processPort"
            operation="call_Announce_Issues_Derived_Process" variable="processData"
             createInstance="Yes"/>
    <invoke name="AnnounceIssuesforVote" partnerLink="WGVoter" portType="tns:emailPort"
            operation="sendVoteAnnouncement" inputVariable="processData"/>
    <invoke name="Collect_Votes_Derived_Process" partnerLink="Internal"
            portType="tns:processPort" operation="call_Collect_Votes_Derived_Process"
            inputVariable="processData"  outputVariable="processData"/>
    <reply partnerLink="Internal" portType="tns:processPort"
          operation="call_Announce_Issues_Derived_Process"
          variable="processData" createInstance="Yes"/>
  </sequence>
</process>

<!-- A Derived Process -->
<process name="Collect_Votes_Derived_Process">
  <!--this calls the second Sub-Process. After the Collect Votes Sub-Process
      times out, the rest of the process will be in the fault handler
      of that process. Calls from there will loop back into other processes.-->
  <variables>
    <variable name="processData" messageType="processDataMessage"/>
  </variables>
  <sequence>
    <receive partnerLink="Internal" portType="tns:processPort"
            operation="call_Collect_Votes_Derived_Process" variable="processData"
            createInstance="Yes"/>
    <invoke name="Collect_Votes" partnerLink="Internal" portType="tns:processPort"
          operation="call_Collect_Votes" inputVariable="processData"
            outputVariable="processData"/>
```

```xml
      <reply partnerLink="Internal" portType="tns:processPort"
            operation="call_Collect_Votes_Derived_Process" variable="processData"
            createInstance="Yes"/>
   </sequence>
</process>
<!-- A Derived Process -->
<process name="Issues_wo_Majority_Derived_Process">
   <variables>
      <variable name="processData" messageType="processDataMessage"/>
   </variables>
   <sequence>
      <receive partnerLink="Internal" portType="tns:processPort"
            operation="call_Issues_wo_Majority_Derived_Process" variable="processData"
             createInstance="Yes"/>
      <switch name="IssueswoMajority">
        <case name="Yes"
             condition="bpws:getVariableProperty(ProcessData,NoMajority)=true">
           <switch name="2ndTime">
             <!-- name="Yes" -->
             <case condition="bpws:getVariableProperty(ProcessData,VotedOnce)=true">
               <!--This is done to do the complex looping situation. -->
               <invoke name="Discussion_Cycle_Derived_Process" partnerLink="Internal"
                    portType="tns:processPort"
                     operation="call_Discussion_Cycle_Derived_Process"
                    inputVariable="processData" outputVariable="processData"/>
             </case>
             <!-- name="No (otherwise)" -->
             <otherwise>
               <sequence>
                 <flow>
                   <invoke name="ReducetoTwoSolutions" partnerLink="internal"
                        portType="tns:internalPort" operation="sendReceiveSolutions"
                        inputVariable="processData" outputVariable="processData"/>
                   <invoke name="EMail Voters that have to Change Votes"
                         partnerLink="WGVoter" portType="tns:emailPort"
                         operation="sendVoteWarning" inputVariable="processData"/>
                 </flow>
                 <invoke process="Announce_Issues_Derived_Process" partnerLink="Internal"
                       portType="tns:processPort"
                        operation="call_Announce_Issues_Derived_Process"
                       inputVariable="processData" outputVariable="processData"/>
               </sequence>
             </otherwise>
           </switch>
        </case>
        <otherwise name="Nootherwise">
           <!-- This is one of the two ways to the end of the Process. -->
           <empty/>
        </otherwise>
```

```
      </switch>
    </sequence>
</process>
<!-- A User Built Process -->
<process name="Discussion_Cycle">
  <!--This defines the first Sub-Process. -->
  <variables>
    <variable name="processData" messageType="processDataMessage"/>
  </variables>
  <sequence>
    <receive partnerLink="Internal" portType="tns:processPort"
             operation="call_Discussion_Cycle" variable="processData"
             createInstance="Yes"/>
    <invoke name="AnnounceIssuesforDiscussion" partnerLink="WGVoter"
            portType="tns:emailPort" operation="sendDiscussionAnnouncement"
            inputVariable="processData"/>
    <flow>
      <links>
        <link name="CheckCalendarforConferenceCalltoWaituntilThursday9am"/>
        <link name="CheckCalendarforConferenceCalltoEmpty"/>
        <link name="WaituntilThursday9amtoModerateConferenceCallDiscussion"/>
      </links>
      <!-- This is the first of the three paths of the fork. -->
      <scope>
        <invoke name="ModerateEmailDiscussion" partnerLink="internal"
                portType="tns:internalPort" operation="sendDiscussion"
                inputVariable="processData" outputVariable="processData"/>
        <faultHandlers>
          <catch faultName="7Days_Exit">
            <empty/>
          </catch>
        </faultHandlers>
        <eventHandlers>
          <onAlarm for="tns:OneWeek">
            <throw faultName="7Days_Exit"/>
          </catch>
        </eventHandlers>
      </scope>
      <!-- This is the second of the three paths of the fork. -->
      <sequence>
        <wait name="Delay6daysfromDiscussionAnnouncement" for="P6D"/>
        <invoke name="EMailDiscussionDeadlineWarning" partnerLink="WGVoter"
                portType="tns:emailPort" operation="sendDiscussionWarning"
                inputVariable="processData">
        </invoke>
      </sequence>
      <!-- This is the third of the three paths of the fork. -->
      <invoke name="CheckCalendarforConferenceCall" partnerLink="internal"
              portType="tns:internalPort" operation="receiveCallSchedule"
```

```
              inputVariable="processData" outputVariable="processData">
          <source linkName="CheckCalendarforConferenceCalltoWaituntilThursday9am"
              transitionCondition="bpws:getVariableProperty(processData,conCall)=true"/>
          <source linkName="CheckCalendarforConferenceCalltoEmpty"
            transitionCondition="not(bpws:getVariableProperty(processData,conCall)=true)"/>
        </invoke>
        <!-- name="Yes" -->
        <wait name="WaituntilThursday9am" for="P6DT9H">
          <target linkName=
                 "CheckCalendarforConferenceCalltoWaituntilThursday9am">
          <source linkName="WaituntilThursday9amtoModerateConferenceCallDiscussion"/>
        </wait>
        <invoke name="ModerateConferenceCallDiscussion" partnerLink="internal"
              portType="tns:internalPort" operation="sendConCall"
              inputVariable="processData" outputVariable="processData">
          <target linkName="WaituntilThursday9amtoModerateConferenceCallDiscussion"/>
        </invoke>
        <!-- name="otherwise" -->
        <empty>
          <target linkName="CheckCalendarforConferenceCalltoEmpty"/>
        </empty>
      </flow>
      <invoke name="EvaluateDiscussionProgress" partnerLink="internal"
            portType="tns:internalPort" operation="receiveDiscussionStatus"
            inputVariable="processData" outputVariable="processData"/>
      <reply partnerLink="Internal" portType="tns:processPort"
            operation="call_Discussion_Cycle" variable="processData"/>
    </sequence>
</process>

<!-- A User Built Process -->
<process name="Collect_Votes">
  <!--This is a process for the E-Mail Voting collection. It consists of an all and a
      timeout event handler. The all will never complete normally since there is an
      infinite loop inside. The timeout is intended to be the normal way of ending the
      process. -->
  <variables>
    <variable name="processData" messageType="processDataMessage"/>
  </variables>
  <sequence>
    <receive partnerLink="Internal" portType="tns:processPort"
          operation="call_Collect_Votes" variable="processData" createInstance="Yes"/>
    <scope>
      <flow>
        <links>
          <link name="Delay6daysfromVoteAnnouncementtoEMailVoteDeadlineWarning"/>
          <link name="CheckCalendarforConferenceCalltoWaituntilThursday9am"/>
          <link name="CheckCalendarforConferenceCalltoEmpty"/>
          <link name="WaituntilThursday9amtoModerateConferenceCallDiscussion"/>
```

```xml
    </links>
    <!--This is the first of the four paths of the fork. -->
    <invoke name="CheckCalendarforConferenceCall" partnerLink="internal"
          portType="tns:internalPort" operation="receiveCallSchedule"
          inputVariable="processData" outputVariable="processData">
      <target linkName="CheckCalendarforConferenceCalltoWaituntilThursday9am"
        transitionCondition="bpws:getVariableProperty(processData,conCall)=true"/>
      <target linkName="CheckCalendarforConferenceCalltoEmpty"
        transitionCondition="not(bpws:getVariableProperty(processData,conCall)=true)"/>
    </invoke>
  <!-- name="Yes" -->
    <wait name="WaituntilThursday9am" for="P6DT9H">
      <source linkName=
            "CheckCalendarforConferenceCalltoWaituntilThursday9am">
      <target linkName="WaituntilThursday9amtoModerateConferenceCallDiscussion"/>
    </wait>
    <invoke name="ModerateConferenceCallDiscussion" partnerLink="internal"
          portType="tns:internalPort" operation="sendConCall"
          inputVariable="processData" outputVariable="processData">
      <source linkName="WaituntilThursday9amtoModerateConferenceCallDiscussion"/>
    </invoke>
    <!-- name="otherwise" -->
    <empty>
      <source linkName="CheckCalendarforConferenceCalltoEmpty"/>
    </empty>
    <!-- This is the second of the four paths of the fork. -->
    <invoke name="ModerateEMailDiscussion" partnerLink="internal"
          portType="tns:internalPort" operation="sendDiscussion"
          inputVariable="processData" outputVariable="processData"/>
    <!--This is the third of the four paths of the fork.-->
    <wait name="Delay6daysfromVoteAnnouncement" for="P6D">
      <target linkName="Delay6daysfromVoteAnnouncementtoEMailVoteDeadlineWarning"/>
    </wait>
    <invoke name="EMailVoteDeadlineWarning" partnerLink="WGVoter"
          portType="tns:emailPort" operation="sendVoteWarning"
          inputVariable="processData">
      <source linkName="Delay6daysfromVoteAnnouncementtoEMailVoteDeadlineWarning"/>
    </invoke>
    <!--This is the fourth of the four paths of the fork. This branch of the all is
        intended to be an infinite loop that is eventually interrupted by the Time
        Out. This is necessary since any voter can change their vote until the
        deadline. -->
    <while condition="1=0">
      <sequence>
        <receive name="ReceiveVote" partnerLink="WGVoter" portType="tns:emailPort"
              operation="receiveVote" variable="processData"/>
        <invoke name="IncrementTally" partnerLink="internal"
              portType="tns:internalPort" operation="sendReceiveTotal"
              inputVariable="processData" outputVariable="processData"/>
```

```xml
            </sequence>
          </while>
        </flow>
        <eventHandlers>
          <onAlarm for="P7D">
            <throw faultName="7days_Exit"/>
          </onAlarm>
        </eventHandlers>
        <faultHandlers>
          <catch faultName="7days_Exit">
            <!-- The BPMN Diagram shows that the Timer Intermediate Event connects
                 directly to the rest of the Process. Thus, they will show up in
                 this activity set. -->
            <sequence>
            <invoke name="PrepareResults" partnerLink="internal"
                    portType="tns:internalPort" operation="sendReceiveResults"
                    inputVariable="processData" outputVariable="processData"/>
            <flow>
              <invoke name="PostResultsonWebSite" partnerLink="internal"
                    portType="tns:internalPort" operation="postVotingResults"
                    inputVariable="processData"/>
              <invoke name="EMailResultsofVote" partnerLink="WGVoter"
                    portType="tns:emailPort" operation="sendVotingResults"
                    inputVariable="processData"/>
            </flow>
            <switch name="DidEnoughMembersVote">
              <!-- name="No" -->
              <case condition="bpws:getVariableProperty(ProcessData,NumVoted)>
                          (.7)*(bpws:getVariableProperty(ProcessData,NumVWGM))">
                <switch name="Havethemembersbeenwarned">
                  <!-- name="Yes" -->
                  <case condition="bpws:getVariableProperty(ProcessData,
                               VotersWarned)=true">
                    <sequence>
                      <invoke name="ReducenumberofVotingMembersandRecalculateVote"
                            partnerLink="internal" portType="tns:internalPort"
                            operation="sendReceiveNumVoters" inputVariable="processData"
                            outputVariable="processData"/>
                      <!--Some elements of the process were separated into a derived process
                            since they would have been repeated. They would have been
                            repeated because they are arrived by alternativepaths that do not
                            close a set of alternative paths. -->
                      <invoke name="Issues_wo_Majority_Derived_Process" partnerLink="Internal"
                            PortType="tns:processPort"
                                operation="call_Issues_wo_Majority_Derived_Process"
                            inputVariable="processData" outputVariable="processData"/>
                    </sequence>
                  </case>
                  <!-- name="No (otherwise)" -->
```

```xml
            <otherwise>
              <sequence>
                <invoke name="ReannounceVotewithwarningtovotingmembers"
                        partnerLink="WGVoter" portType="tns:emailPort"
                        operation="sendReannounceVote" inputVariable="processData"
                        outputVariable="processData"/>
                <invoke name="Collect_Votes_Derived_Process" partnerLink="Internal"
                         portType="tns:processPort"
                        operation="call_Collect_Votes_Derived_Process"
                        inputVariable="processData" outputVariable="processData"/>
              </sequence>
            </otherwise>
          </switch>
        </case>
      <!-- name="Yes (otherwise)" -->
        <otherwise>
          <!-- Some elements of the process were separated into a derived process
               since they would have been repeated. They would have been repeated
               because they are arrived by alternative that do not close a set of
               alternative paths. -->
          <invoke process="Issues_wo_Majority_Derived_Process" partnerLink="Internal"
                  portType="tns:processPort"
                     operation="call_Issues_wo_Majority_Derived_Process"
                  inputVariable="processData" outputVariable="processData"/>
        </otherwise>
      </switch>
    </sequence>
   </catch>
  </faultHandlers>
 </scope>
 <reply partnerLink="Internal" portType="tns:processPort"
        operation="call_Collect_Votes" variable="processData"/>
 </sequence>
</process>
```

Annex B

(informative)

BPMN Element Attributes and Types

This annex provides the complete set of BPMN Element Attributes and the definition of types that support the Attributes. All the tables in this annex also appear in Chapters 8, 9, and 10.

The following figure displays a diagram of the relationship between the main BPMN Event elements and their attributes (see Figure B.1). Other diagrams in this Annex will provide more detailed information about specific groups of elements (e.g, Events and their related elements and attributes).
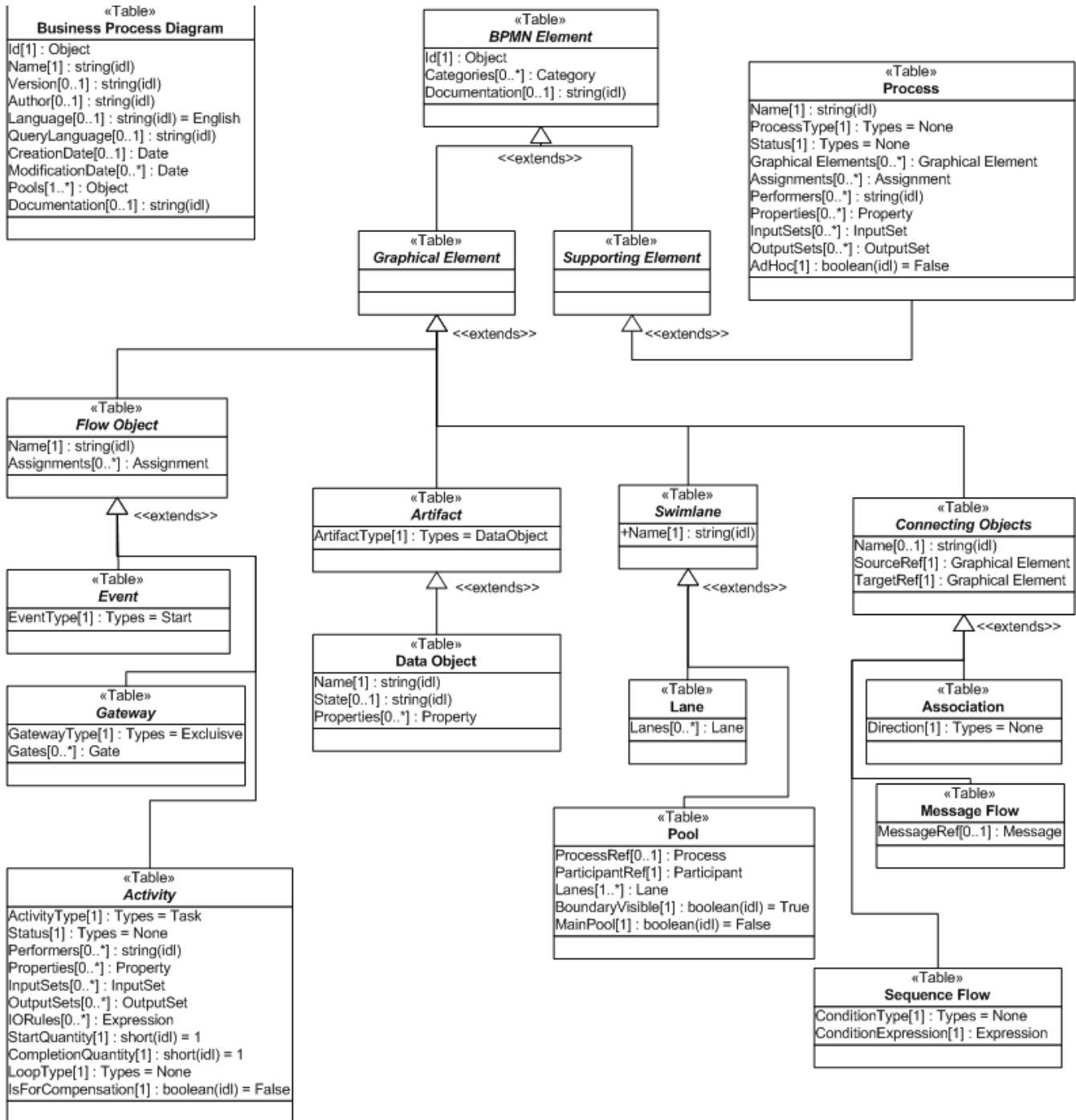
Figure B.1 - Main BPMN Elements and Attributes

B.1 Business Process Diagram Attributes

The following table displays the set of attributes of a Business Process Diagram:

Table B.48 - Business Process Diagram Attributes

Attributes	Description
Id : Object	This is a unique Id that distinguishes the Diagram from other Diagrams.
Name : String	Name is an attribute that is text description of the Diagram.
Version (0-1) : String	This defines the Version number of the Diagram.
Author (0-1) : String	This holds the name of the author of the Diagram.
Language (0-1) : String	This holds the name of the language in which text is written. The default is English.
QueryLanguage (0-1) : String	A Language MAY be provided so that the syntax of queries used in the Diagram can be understood.
CreationDate (0-1) : Date	This defines the date on which the Diagram was create (for this Version).
ModificationDate (0-1) : Date	This defines the date on which the Diagram was last modified (for this Version).
Pools (1-n) : Pool	A BPD SHALL contain one or more Pools. The boundary of one of the Pools MAY be invisible (especially if there is only one Pool in the Diagram). Refer to "Pool" on page 75 for more information about Pools.
Documentation (0-1) : String	The modeler MAY add optional text documentation about the Diagram.

B.2 BPMN Elements

B.2.1 Common BPMN Element Attributes

The following table displays a set of common attributes for BPMN elements (graphical elements and supporting elements).

Table B.49 - Common BPMN Element Attributes

Attributes	Description
Id : Object	This is a unique Id that identifies the object from other objects within the Diagram.
Categories (0-n) : Category	The modeler MAY add one or more defined Categories, which have user-defined semantics, and that can be used for purposes such as reporting and analysis. The details of Catogories is defined in "Category on page 277."
Documentation (0-1) : String	The modeler MAY add text documentation about the object.

These attributes are used for Graphical Elements [which are Flow Objects (Section B.4, "Common Flow Object Attributes," on page 250), Connecting Objects (Section B.10, "Graphical Connecting Objects," on page 271), Swimlanes (Section B.8, "Swimlanes (Pools and Lanes)," on page 267), and Artifacts (Section B.9, "Artifacts," on page 269)], and Supporting Elements (Section B.11, "Supporting Elements," on page 274).

B.2.2 Graphical Elements

Graphical Element is one of two main elements that are of type BPMN Element (see Figure B.1). The other is Supporting Element. There are four main types, and many subtypes, of Graphical Elements. These are: Artifacts (see Section B.9, "Artifacts," on page 269), Connecting Objects (see Section B.10, "Graphical Connecting Objects," on page 271), Flow Objects (see Section B.4, "Common Flow Object Attributes," on page 250), and Swimlanes (see Section B.8, "Swimlanes (Pools and Lanes)," on page 267).

B.2.3 Supporting Elements

Supporting Element (see Section B.11, "Supporting Elements," on page 274) is one of two main elements that are of type BPMN Element (see Figure B.1). The other is Graphical Element.

B.3 Process Attributes

The following table displays the set of attributes of a Process, and which extends the set of common BPMN Element attributes (see Table B.49):

Table B.50 - Process Attributes

Attributes	Description
Name : String	Name is an attribute that is a text description of the object.
ProcessType (None | Private | Abstract | Collaboration) None : String	ProcessType is an attribute that provides information about which lower-level language the Pool will be mapped. By default, the ProcessType is None (or undefined).
Status (None | Ready | Active | Cancelled | Aborting | Aborted | Completing | Completed) None : String	The Status of a Process is determined when the Process is being executed by a process engine. The Status of a Process can be used within Assignment Expressions.
GraphicalElements (0-n) : Object	The GraphicalElements attribute identifies all of the objects (e.g., Events, Activities, Gateways, and Artifacts) that are contained within the Process.
Assignments (0-n) : Assignment	One or more assignment expressions MAY be made for the object. The Assignment SHALL be performed as defined by the AssignTime attribute (see below). The details of Assignment is defined in "Assignment on page 277."
Performers (0-n) : String	One or more Performers MAY be entered. The Performers attribute defines the resource that will be responsible for the Process. The Performers entry could be in the form of a specific individual, a group, an organization role or position, or an organization.

Table B.50 - Process Attributes

Attributes	Description
Properties (0-n) : Property	Modeler-defined Properties MAY be added to a Process. These Properties are "local" to the Process. All Tasks, Sub-Process objects, and Sub-Processes that are embedded SHALL have access to these Properties. The fully delineated name of these properties is "<process name>.<property name>" (e.g., "Add Customer.Customer Name"). If a process is embedded within another Process, then the fully delineated name SHALL also be preceded by the Parent Process name for as many Parents there are until the top level Process. Further details about the definition of a Property can be found in "Property on page 285."
InputSets (0-n) : InputSet	The InputSets attribute defines the data requirements for input to the Process. Zero or more InputSets MAY be defined. Each Input set is sufficient to allow the Process to be performed (if it has first been instantiated by the appropriate signal arriving from an incoming Sequence Flow). Further details about the definition of an Input-Set can be found in Section B.11.10, "InputSet," on page 282.
OutputSets (0-n) : OutputSet	The OutputSets attribute defines the data requirements for output from the Process. Zero or more OutputSets MAY be defined. At the completion of the Process, only one of the OutputSets may be produced--It is up to the implementation of the Process to determine which set will be produced. However, the IORules attribute MAY indicate a relationship between an OutputSet and an InputSet that started the Process. Further details about the definition of an OutputSet can be found in Section B.11.13, "OutputSet," on page 284.
AdHoc False : Boolean	AdHoc is a boolean attribute, which has a default of False. This specifies whether the Process is Ad Hoc or not. The activities within an Ad Hoc Process are not controlled or sequenced in a particular order, their performance is determined by the performers of the activities. If set to True, then the Ad Hoc marker SHALL be placed at the bottom center of the Process or the Sub-Process shape for Ad Hoc Processes.
[AdHoc = True only] **AdHocOrdering** (0-1) (Sequential | Parallel) Parallel : String	If the Process is Ad Hoc (the AdHoc attribute is True), then the AdHocOrdering attribute MUST be included. This attribute defines if the activities within the Process can be performed in Parallel or must be performed sequentially. The default setting is Parallel and the setting of Sequential is a restriction on the performance that may be required due to shared resources.
[AdHoc = True only] **AdHocCompletionCondition** (0-1) : Expression	If the Process is Ad Hoc (the AdHoc attribute is True), then the AdHocCompletionCondition attribute MUST be included. This attribute defines the conditions when the Process will end.

B.4 Common Flow Object Attributes

The following table displays the set of attributes common to BPMN Flow Objects (Events, Activities, and Gateways), and which extends the set of common BPMN Element attributes (see Table B.49).

Table B.51 - Common Flow Object Attributes

Attributes	Description Figure
Name : String	Name is an attribute that is a text description of the object.
Assignments (0-n) : Assignment	One or more assignment expressions MAY be made for the object. For activities, the Assignment SHALL be performed as defined by the AssignTime attribute. The Details of the Assignment is defined in "Assignment on page 277."

B.5 Events

The following figure displays a diagram of the relationship between BPMN Event elements and their attributes (see Figure B.2).
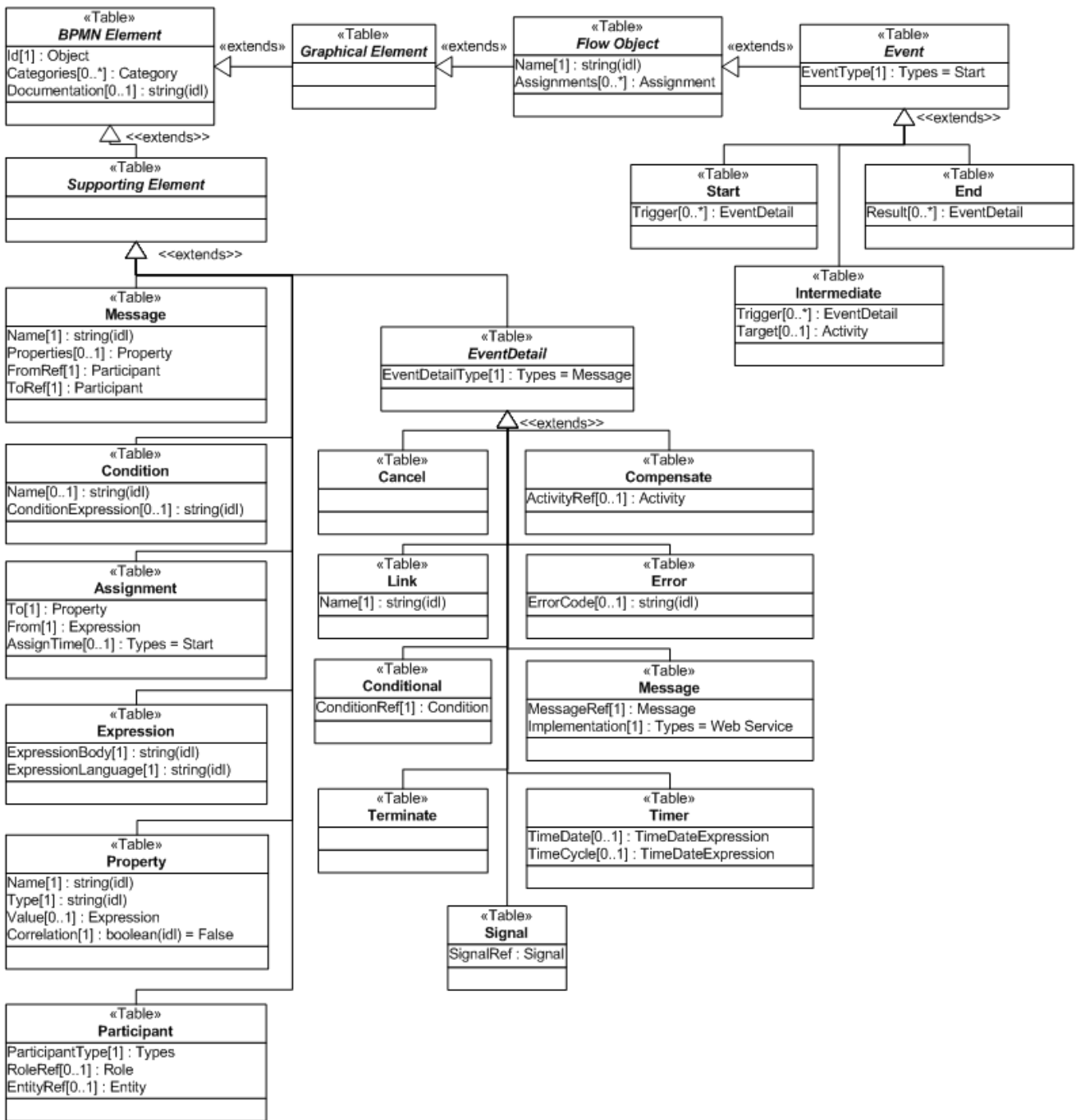
Figure B.2 - BPMN Event Elements and Attributes

B.5.1　Common Event Attributes

The following table displays the set of attributes common to the three types of Events, and which extends the set of common Flow Object attributes (see Table B.51).

Table B.52 - Common Event Attributes

Attributes	Description
EventType (Start | End | Intermediate) Start : String	The EventType MUST be of type Start, End, or Intermediate.

B.5.2　Start Event

The following table displays the set of attributes of a Start Event, which extends the set of common Event elements (see Table B.52).

Table B.53 - Start Event Attributes

Attributes	Description
Trigger (0-n) : EventDetail	Trigger (EventDetail) is an attribute that defines the type of trigger expected for a Start Event. Of the set of EventDetailTypes (see Section B.11.7, "Event Details," on page 278), only four (4) can be applied to a Start Event: Message, Timer, Conditional, and Signal (see Table 9.4). If there is no EventDetail is defined, then this is considered a None End Event and the Event will not have an internal marker (see Table 9.4). If there is more than one EventDetail is defined, this is considered a Multiple End Event and the Event will have the star internal marker (see Table 9.4).

B.5.3　End Event

The following table displays the set of attributes of a End Event, which extends the set of common Event elements (see Table B.52).

Table B.54 - End Event Attributes

Attributes	Description
Result (0-n) : EventDetail	Result (EventDetail) is an attribute that defines the type of result expected for an End Event. Of the set of EventDetailTypes (see Section B.11.7, "Event Details," on page 278), only six (6) can be applied to an End Event: Message, Error, Cancel, Compensation, Signal, and Terminate (see Table 9.6). If there is no EventDetail is defined, then this is considered a None End Event and the Event will not have an internal marker (see Table 9.6). If there is more than one EventDetail is defined, this is considered a Multiple End Event and the Event will have the star internal marker (see Table 9.6).

B.5.4 Intermediate Event

The following table displays the set of attributes of an Intermediate Event, which extends the set of common Event elements (see Table B.52).

Table B.55 - Intermediate Event Attributes

Attributes	Description
Trigger (0-n) : EventDetail	Trigger (EventDetail) is an attribute that defines the type of trigger expected for an Intermediate Event. Of the set of EventDetailTypes (see Section B.11.7, "Event Details," on page 278), only eight (8) can be applied to an Intermediate Event: Message, Timer, Error, Cancel, Compensation, Conditional, Link, and Signal. (see Table 9.8). If there is no EventDetail is defined, then this is considered a None Intermediate Event and the Event will not have an internal marker (see Table 9.8). If there is more than one EventDetail is defined, this is considered a Multiple Intermediate Event and the Event will have the star internal marker (see Table 9.8).
Target (0-1) : Activity	A Target MAY be included for the Intermediate Event. The Target MUST be an activity (Sub-Process or Task). This means that the Intermediate Event is attached to the boundary of the activity and is used to signify an exception or compensation for that activity.

B.6   Activities

The following figure displays a diagram of the relationship between BPMN activity elements and their attributes (see Figure B.3).
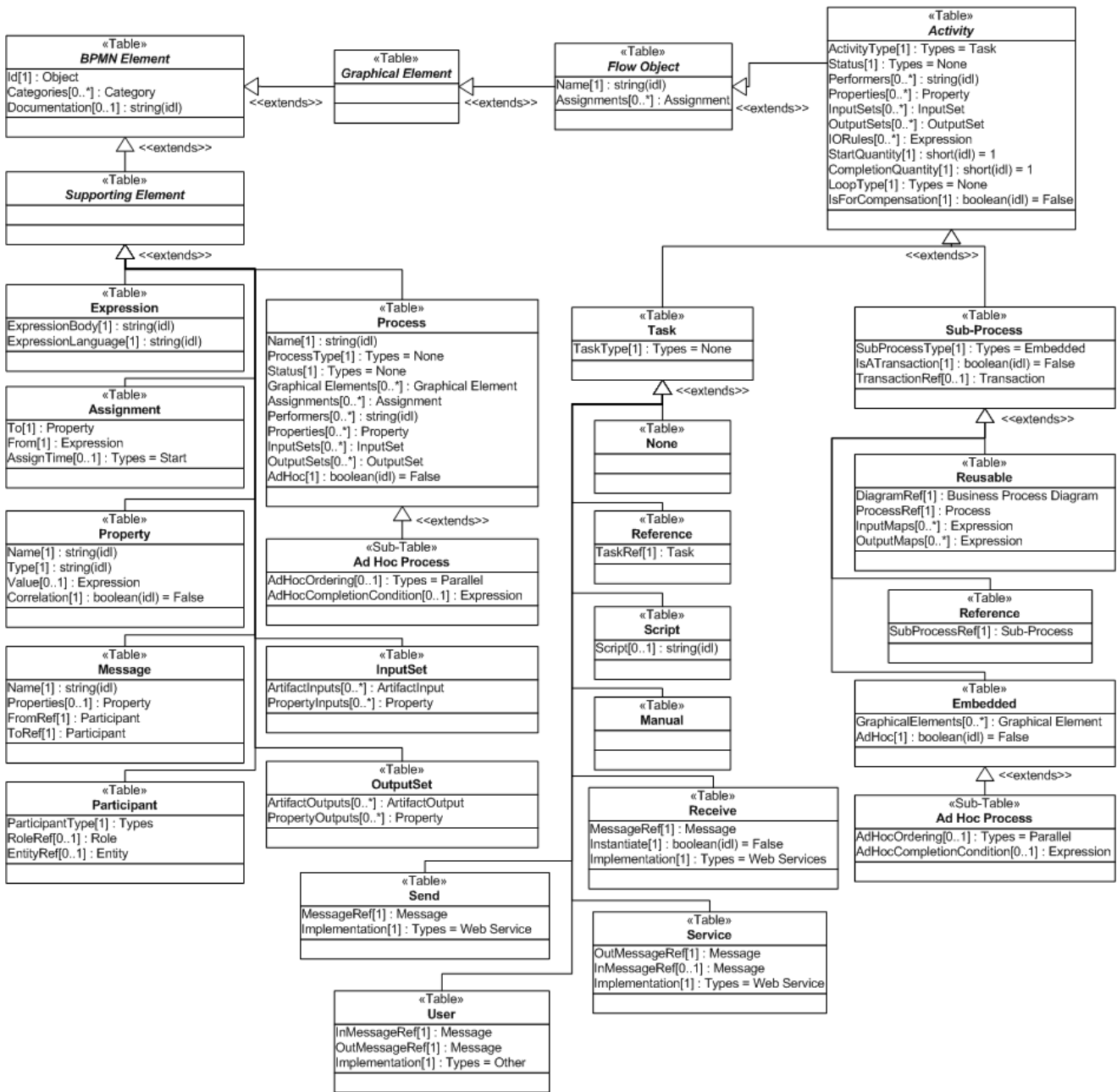
Figure B.3 - BPMN Activity Elements and Attributes

B.6.1 Common Activity Attributes

The following table displays the set of attributes common to both a Sub-Process and a Task, and which extends the set of common Flow Object attributes (see Table B.51) -- Note that Figure 10.55 and Figure 10.56 contain additional attributes that must be included within this set if extended by any other attribute table.

Table B.56 - Common Activity Attributes

Attributes	Description
ActivityType (Task | Sub-Process) Task : String	The ActivityType MUST be of type Task or Sub-Process.
Status (None | Ready | Active | Cancelled | Aborting | Aborted | Completing | Completed) None : String	The Status of an activity is determined when the activity is being executed by a process engine. The Status of an activity can be used within Assignment Expressions.
Performers (0-n) : String	One or more Performers MAY be entered. The Performer attribute defines the resource that will perform or will be responsible for the activity. The Performer entry could be in the form of a specific individual, a group, an organization role or position, or an organization.
Properties (0-n) : Property	Modeler-defined Properties MAY be added to an activity. These Properties are "local" to the activity object. These Properties are only for use within the processing of the activity. The fully delineated name of these properties are "<process name>.<sub-process name>.<property name>" (e.g., "Add Customer.Review Credit.Status"). Further details about the definition of a Property can be found in "Property on page 285."
InputSets (0-n) : InputSet	The InputSets attribute defines the data requirements for input to the activity. Zero or more InputSets MAY be defined. Each Input set is sufficient to allow the activity to be performed (if it has first been instantiated by the appropriate signal arriving from an incoming Sequence Flow). Further details about the definition of an Input-Set can be found in Section B.11.10, "InputSet," on page 282.
OutputSets (0-n) : OutputSet	The OutputSets attribute defines the data requirements for output from the activity. Zero or more OutputSets MAY be defined. At the completion of the activity, only one of the OutputSets may be produced--It is up to the implementation of the activity to determine which set will be produced. However, the IORules attribute MAY indicate a relationship between an OutputSet and an InputSet that started the activity. Further details about the definition of an OutputSet can be found in Section B.11.13, "OutputSet," on page 284.[
IORules (0-n) : Expression	The IORules attribute is a collection of expressions, each of which specifies the required relationship between one input and one output. That is, if the activity is instantiated with a specified input, that activity shall complete with the specified output.

Table B.56 - Common Activity Attributes

Attributes	Description
StartQuantity 1 : Integer	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of Tokens that must arrive before the activity can begin.
CompletionQuantity 1 : Integer	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of Tokens that must be generated from the activity. This number of Tokens will be sent done any outgoing Sequence Flow (assuming any Sequence Flow Conditions are satisfied).
LoopType (None | Standard | MultiInstance) None : String	LoopType is an attribute and is by default None, but MAY be set to Standard or MultiInstance. If so, the Loop marker SHALL be placed at the bottom center of the activity shape (see Figure 9.6 and Figure 9.15). A Task of type Receive that has its Instantiate attribute set to True MUST NOT have a Standard or MultiInstance LoopType.

Standard Loop Attributes

The following are additional attributes of a Standard Loop Activity (where the LoopType attribute is set to "Standard"), which extends the set of common activity attributes (see Table B.56).

Table B.57 - Standard Loop Activity Attributes

Attributes	Description
LoopCondition : Expression	Standard Loops MUST have a boolean Expression to be evaluated, plus the timing when the expression SHALL be evaluated. The attributes of an Expression can be found in "Expression on page 281."
LoopCounter : Integer	The LoopCounter attribute is used at runtime to count the number of loops and is automatically updated by the process engine. The LoopCounter attribute MUST be incremented at the start of a loop. The modeler may use the attribute in the LoopCondition Expression.
LoopMaximum (0-1) : Integer	The Maximum an optional attribute that provides is a simple way to add a cap to the number of loops. This SHALL be added to the Expression defined in the LoopCondition.
TestTime (Before | After) After : String	The expressions that are evaluated Before the activity begins are equivalent to a programming while function. The expression that are evaluated After the activity finishes are equivalent to a programming until function.

Multi-Instance Loop Attributes

The following are additional attributes of a Multi-Instance Loop Activity (where the LoopType attribute is set to "MultiInstance"), which extends the set of common activity attributes (see Table B.56).

Table B.58 - Multi-Instance Loop Activity Attributes

Attributes	Description
MI_Condition : Expression	MultiInstance Loops MUST have a numeric Expression to be evaluated--the Expression MUST resolve to an integer. The attributes of an Expression can be found in "Expression on page 281."
LoopCounter : Integer	The LoopCounter attribute is only applied for Sequential MultiInstance Loops and for processes that are being executed by a process engine. The attribute is updated at runtime by a process engine to count the number of loops as they occur. The LoopCounter attribute MUST be incremented at the start of a loop. Unlike a Standard loop, the modeler does not use this attribute in the MI_Condition Expression, but it can be used for tracking the status of a loop.
MI_Ordering (Sequential | Parallel) Sequential : String	This applies to only MultiInstance Loops. The MI_Ordering attribute defines whether the loop instances will be performed sequentially or in parallel. Sequential MI_Ordering is a more traditional loop. Parallel MI_Ordering is equivalent to multi-instance specifications that other notations, such as UML Activity Diagrams use. If set to Parallel, the Parallel marker SHALL replace the Loop Marker at the bottom center of the activity shape (see Figure 9.9 and Figure 9.15).
[Parallel MI_Ordering only] **MI_FlowCondition** (None | One | All | Complex) All : String	This attribute is equivalent to using a Gateway to control the flow past a set of parallel paths. • An MI_FlowCondition of "None" is the same as uncontrolled flow (no Gateway) and means that all activity instances SHALL generate a token that will continue when that instance is completed. • An MI_FlowCondition of "One" is the same as an Exclusive Gateway and means that the Token SHALL continue past the activity after only one of the activity instances has completed. The activity will continue its other instances, but additional Tokens MUST NOT be passed from the activity. • An MI_FlowCondition of "All" is the same as a Parallel Gateway and means that the Token SHALL continue past the activity after all of the activity instances have completed. • An MI_FlowCondition of "Complex" is similar to that of a Complex Gateway. The ComplexMI_FlowCondition attribute will determine the Token flow.
[Complex MI_FlowCondition only] **ComplexMI_FlowCondition** (0-1) : Expression	If the MI_FlowCondition attribute is set to "Complex," then an Expression Must be entered. This Expression that MAY reference Process data. The expression will be evaluated after each iteration of the Activity and SHALL resolve to a boolean. If the result of the expression evaluation is TRUE, then a Token will be sent down the activity's outgoing Sequence Flow. Otherwise, no Token will be sent. The attributes of an Expression can be found in "Expression on page 281."

B.6.2 Sub-Process

The following table displays the set of attributes of a Sub-Process, which extends the set of common activity attributes (see Table B.56).

Table B.59 - Sub-Process Attributes

Attributes	Description
SubProcessType (Embedded | Reusable | Reference) Embedded : String	SubProcessType is an attribute that defines whether the Sub-Process details are embedded within the higher level Process or refers to another, re-usable Process. The default is Embedded.
IsATransaction False : Boolean	IsATransaction determines whether or not the behavior of the Sub-Process will follow the behavior of a Transaction (see "Sub-Process Behavior as a Transaction on page 63.")
TransactionRef (0-1) : Transaction	If the IsATransaction attribute is False, then a Transaction MUST NOT be identified. If the IsATransaction attribute is True, then a Transaction MUST be identified. The attributes of a Transaction can be found in "Transaction on page 286" Note that Transactions that are in different Pools and are connected through Message Flow MUST have the same TransactionId.

Embedded Sub-Process

The following are additional attributes of a Embedded Sub-Process (where the SubProcessType attribute is set to "Embedded"), which extends the set of Sub-Process attributes (see Table B.59).

Table B.60 - Embedded Sub-Process Attributes

Attributes	Description
GraphicalElements (0-n) : Object	The GraphicalElements attribute identifies all of the objects (e.g., Events, Activities, Gateways, and Artifacts) that are contained within the Embedded Sub-Process.

Table B.60 - Embedded Sub-Process Attributes

Attributes	Description
AdHoc False : Boolean	AdHoc is a boolean attribute, which has a default of False. This specifies whether the Embedded Sub-Process is Ad Hoc or not. The activities within an Ad Hoc Embedded Sub-Process are not controlled or sequenced in a particular order, there performance is determined by the performers of the activities.
[AdHoc = True only] **AdHocOrdering** (0-1) (Sequential | Parallel) Parallel : String	If the Embedded Sub-Process is Ad Hoc (the AdHoc attribute is True), then the AdHocOrdering attribute MUST be included. This attribute defines if the activities within the Process can be performed in Parallel or must be performed sequentially. The default setting is Parallel and the setting of Sequential is a restriction on the performance that may be required due to shared resources.
[AdHoc = True only] **AdHocCompletionCondition** (0-1) : Expression	If the Embedded Sub-Process is Ad Hoc (the AdHoc attribute is True), then a Completion Condition MUST be included, which defines the conditions when the Process will end. The Ad Hoc marker SHALL be placed at the bottom center of the Process or the Sub-Process shape for Ad Hoc Processes.

Reusable Sub-Process Attributes

The following are additional attributes of a Reusable Sub-Process (where the SubProcessType attribute is set to "Reusable"), which extends the set of Sub-Process attributes (see Table B.59).

Table B.61 - Reusable Sub-Process Attributes

Attributes	Description
DiagramRef : Business Process Diagram	The BPD MUST be identified. The attributes of a BPD can be found in "Business Process Diagram Attributes on page 31."
ProcessRef : Process	A Process MUST be identified. The attributes of a Process can be found in "Processes on page 32"
InputMaps (0-n) : Expression	Multiple input mappings MAY be made between the Reusable Sub-Process and the Process referenced by this object. These mappings are in the form of an expression. A specific mapping expression MUST specify the mapping of Properties between the two Processes OR the mapping of Artifacts between the two Processes.
OutputMaps (0-n) : Expression	Multiple output mappings MAY be made between the Reusable Sub-Process and the Process referenced by this object. These mappings are in the form of an expression. A specific mapping expression MUST specify the mapping of Properties between the two Processes OR the mapping of Artifacts between the two Processes.

Reference Sub-Process Attributes

The following table displays the set of attributes of a Reference Sub-Process (where the SubProcessType attribute is set to "Reference"), which extends the set of Sub-Process attributes (see Table B.59).

Table B.62 - Reference Sub-Process Attributes

Attributes	Description
SubProcessRef : Sub-Process	The Sub-Process being referenced MUST be identified. The attributes for the Sub-Process element can be found in Table B.59 .

B.6.3   Task

The following table displays the set of attributes of a Task, which extends the set of common activity object attributes (see Table B.56).

Table B.63 - Task Attributes

Attributes	Description
TaskType (Service | Receive | Send | User | Script | Abstract | Manual | Reference | None) None : String	TaskType is an attribute that has a default of None, but MAY be set to Send, Receive, User, Script, Abstract, Manual, Reference, or Service. The TaskType will be impacted by the Message Flow to and/or from the Task, if Message Flow are used. A TaskType of Receive MUST NOT have an outgoing Message Flow. A TaskType of Send MUST NOT have an incoming Message Flow. A TaskType of Script or Manual MUST NOT have an incoming or an outgoing Message Flow. The TaskType list MAY be extended to include new types. The attributes for specific settings of TaskType can be found in Table B.64 through Table B.69.

Service Task Attributes

The following table displays the set of attributes of a Service Task (where the TaskType attribute is set to "Service"), which extends the set of Task attributes (see Table B.63).

Table B.64 - Service Task Attributes

Attributes	Description
InMessageRef : Message	A Message for the InMessageRef attribute MUST be entered. This indicates that the Message will be received at the start of the Task, after the availability of any defined InputSets. One or more corresponding incoming Message Flows MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all incoming Message Flow, but can arrive for only one of the incoming Message Flow for a single instance of the Task.
OutMessageRef : Message	A Message for the OutMessageRef attribute MUST be entered. The sending of this message marks the completion of the Task, which may cause the production of an OutputSet. One or more corresponding outgoing Message Flow MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all outgoing Message Flow and the Message will be sent down all outgoing Message Flow at the completion of a single instance of the Task.
Implementation (Web Service | Other | Unspecified) Web Service : String	This attribute specifies the technology that will be used to send and receive the messages. A Web service is the default technology.

Receive Task Attributes

The following table displays the set of attributes of a Receive Task (where the TaskType attribute is set to "Receive"), which extends the set of Task attributes (see Table B.63).

Table B.65 - Receive Task Attributes

Attributes	Description
MessageRef : Message	A Message for the MessageRef attribute MUST be entered. This indicates that the Message will be received by the Task. The Message in this context is equivalent to an *in-only* message pattern (Web service). One or more corresponding incoming Message Flows MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all incoming Message Flow, but can arrive for only one of the incoming Message Flow for a single instance of the Task.
Instantiate False **: Boolean**	Receive Tasks can be defined as the instantiation mechanism for the Process with the Instantiate attribute. This attribute MAY be set to true if the Task is the first activity after the Start Event or a starting Task if there is no Start Event. Multiple Tasks MAY have this attribute set to True.
Implementation (Web Service | Other | Unspecified) Web Service : String	This attribute specifies the technology that will be used to receive the message. A Web service is the default technology.

Send Task Attributes

The following table displays the set of attributes of a Send Task (where the TaskType attribute is set to "Send"), which extends the set of Task attributes (see Table B.63).

Table B.66 - Send Task Attributes

Attributes	Description
MessageRef : Message	A Message for the MessageRef attribute MUST be entered. This indicates that the Message will be sent by the Task. The Message in this context is equivalent to an *out-only* message pattern (Web service). One or more corresponding outgoing Message Flow MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all outgoing Message Flow and the Message will be sent down all outgoing Message Flow at the completion of a single instance of the Task.
Implementation (Web Service | Other | Unspecified) Web Service : String	This attribute specifies the technology that will be used to send the message. A Web service is the default technology.

User Task Attributes

The following table displays the set of attributes of a User Task (where the TaskType attribute is set to "User"), which extends the set of Task attributes (see Table B.63).

Table B.67 - User Task Attributes

Attributes	Description
InMessageRef : Message	A Message for the InMessageRef attribute MUST be entered. This indicates that the Message will be received at the start of the Task, after the availability of any defined InputSets. One or more corresponding incoming Message Flows MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all incoming Message Flow, but can arrive for only one of the incoming Message Flow for a single instance of the Task.
OutMessageRef : Message	A Message for the OutMessageRef attribute MUST be entered. The sending of this message marks the completion of the Task, which may cause the production of an OutputSet. One or more corresponding outgoing Message Flow MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all outgoing Message Flow and the Message will be sent down all outgoing Message Flow at the completion of a single instance of the Task.
Implementation (Web Service | Other | Unspecified) Web Service : String	This attribute specifies the technology that will be used by the Performers to perform the Task. A Web service is the default technology.

Script Task Attributes

The following table displays the set of attributes of a Script Task (where the TaskType attribute is set to "Script"), which extends the set of Task attributes (see Table B.63).

Table B.68 - Script Task Attributes

Attributes	Description
Script (0-1) : String	The modeler MAY include a script that can be run when the Task is performed. If a script is not included, then the Task will act equivalent to a TaskType of None.

Manual Task Attributes

The Manual Task does not contain any additional attributes beyond the set of Task attributes (see Table B.63):

Reference Task Attributes

The following table displays the set of attributes of a Reference Task (where the TaskType attribute is set to "Reference"), which extends the set of Task attributes (see Table B.63).

Table B.69 - Reference Task Attributes

Attributes	Description
TaskRef : Task	The Task being referenced MUST be identified. The attributes for the Task element can be found in Table B.63.

B.7 Gateways

The following figure displays a diagram of the relationship between BPMN Gateway elements and their attributes (see Figure B.4).

Event-Based Gateways can be defined as the instantiation mechanism for the Process with the Instantiate attribute. This attribute MAY be set to true if the Gateway is the first element after the Start Event or a starting Gateway if there is no Start Event (i.e., there are no incoming Sequence Flow).
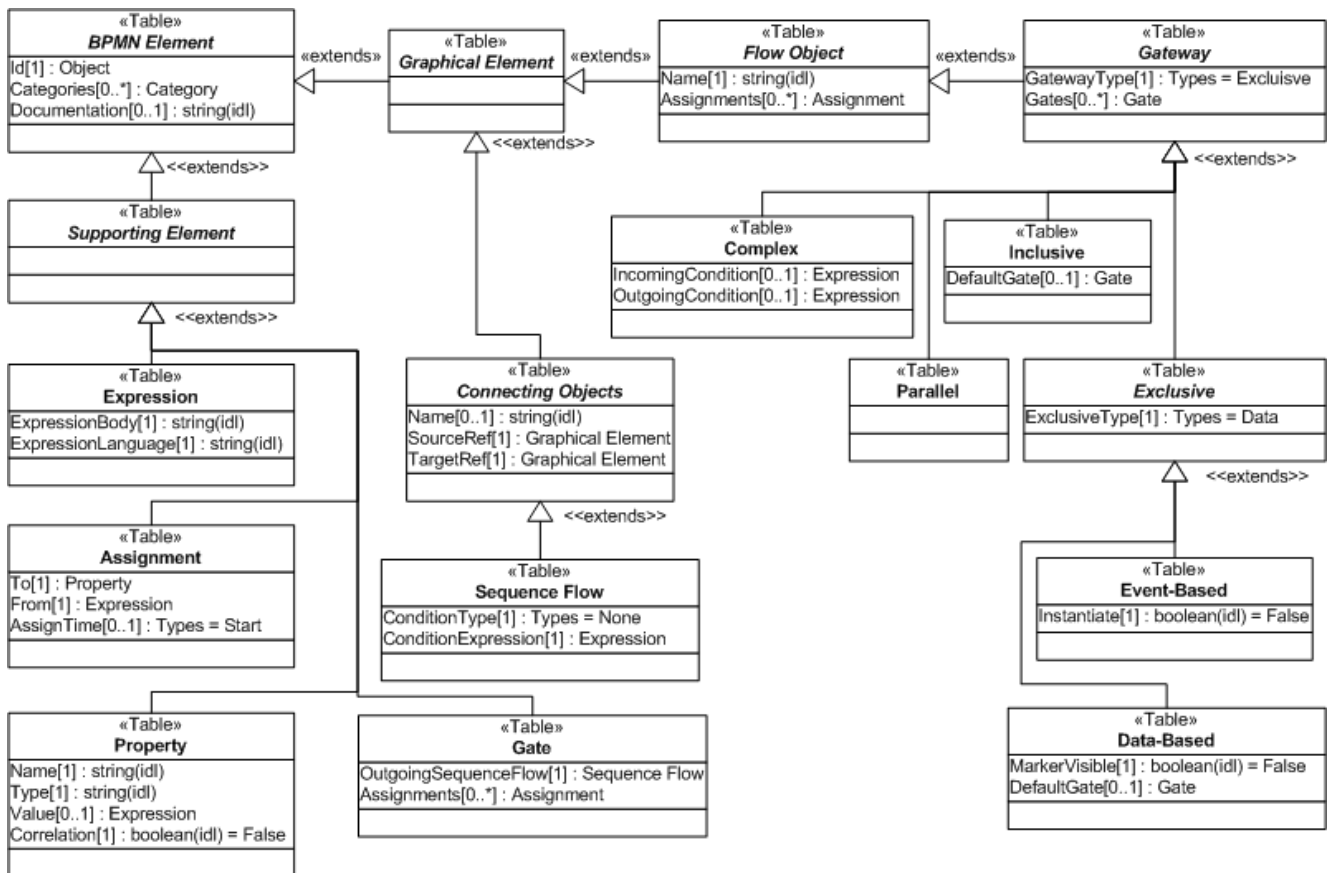
Figure B.4 - BPMN Gateway Elements and Attributes

B.7.1 Common Gateway Attributes

The following table displays the attributes common to Gateways, and which extends the set of common Flow Object attributes (see Table B.51).

Table B.70 - Common Gateway Attributes

Attributes	Description
GatewayType (Exclusive | Inclusive | Complex | Parallel) Exclusive : String	GatewayType is by default Exclusive. The GatewayType MAY be set to Inclusive, Complex, or Parallel. The GatewayType will determine the behavior of the Gateway, both for incoming and outgoing Sequence Flow, and will determine the internal indicator (as shown in Figure 9.17).
Gates (0-n) : Gate	There MAY be zero or more Gates (except where noted below). Zero Gates are allowed if the Gateway is last object in a Process flow and there are no Start or End Events for the Process. If there are zero or only one incoming Sequence Flow, then there MUST be at least two Gates. *For Exclusive Data-Based Gateways* When two Gates are required, one of them MAY be the DefaultGate. *For Exclusive Event-Based Gateways* There MUST be two or more Gates. (Note that this type of Gateway does not act *only* as a Merge--it is always a Decision, at least.) *For Inclusive Gateways* When two Gates are required, one of them MAY be the DefaultGate.

B.7.2 Exclusive Gateways

Data-Based

The following table displays the attributes for an Data-Based Exclusive Gateway. These attributes only apply if the GatewayType attribute is set to Exclusive. The following attributes extend the set of common Gateway attributes (see Table B.70).

Table B.71 - Data-Based Exclusive Gateway Attributes

Attributes	Description
ExclusiveType (Data | Event) Data : String	ExclusiveType is by default Data. The ExclusiveType MAY be set to Event. Since Data-Based Exclusive Gateways is the subject of this section, the attribute MUST be set to Data for the attributes and behavior defined in this section to apply to the Gateway.
MarkerVisible False : Boolean	This attribute determines if the Exclusive Marker is displayed in the center of the Gateway diamond (an "X"). The marker is displayed if the attribute is True and it is not displayed if the attribute is False. By default, the marker is not displayed.
DefaultGate (0-1) : Gate	A Default Gate MAY be specified (see Section B.11.9, "Gate," on page 282).

Event-Based

The following table displays the attributes for an Event-Based Exclusive Gateway. These attributes only apply if the GatewayType attribute is set to Exclusive. The following attributes extend the set of common Gateway attributes (see Table B.70).

Table B.72 - Event-Based Exclusive Gateway Attributes

Attributes	Description
ExclusiveType (Data | Event) Event : String	ExclusiveType is by default Data. The ExclusiveType MAY be set to Event. Since Event-Based Exclusive Gateways is the subject of this section, the attribute MUST be set to Event for the attributes and behavior defined in this section to apply to the Gateway.
Instantiate False : **Boolean**	Event-Based Gateways can be defined as the instantiation mechanism for the Process with the Instantiate attribute. This attribute MAY be set to true if the Gateway is the first element after the Start Event or a starting Gateway if there is no Start Event (i.e., there are no incoming Sequence Flow).

B.7.3 Inclusive Gateways

The following table displays the attributes for an Inclusive Gateway. These attributes only apply if the GatewayType attribute is set to Inclusive. The following attributes extend the set of common Gateway attributes (see Table B.70).

Table B.73 - Inclusive Gateway Attributes

Attributes	Description
DefaultGate (0-1) : Gate	A Default Gate MAY be specified (see Section B.11.9, "Gate," on page 282).

B.7.4 Complex Gateways

The following table displays the attributes for a Complex Gateway. These attributes only apply if the GatewayType attribute is set to Complex. The following attributes extend the set of common Gateway attributes (see Table B.70).

Table B.74 - Complex Gateway Attributes

Attributes	Description
IncomingCondition (0-1) : Expression	If there are Multiple incoming Sequence Flow, an IncomingCondition expression MUST be set by the modeler. This will consist of an expression that can reference Sequence Flow names and or Process Properties (Data).
OutgoingCondition (0-1) : Expression	If there are Multiple outgoing Sequence Flow, an OutgoingCondition expression MUST be set by the modeler. This will consist of an expression that can reference (outgoing) Sequence Flow Ids and or Process Properties (Data).

B.7.5 Parallel Gateways

Parallel Gateways do not have any additional Attributes beyond the common Gateway Attributes (see Table B.70).

B.8 Swimlanes (Pools and Lanes)

The following figure displays a diagram of the relationship between BPMN Swimlane elements and their attributes (see Figure B.5).
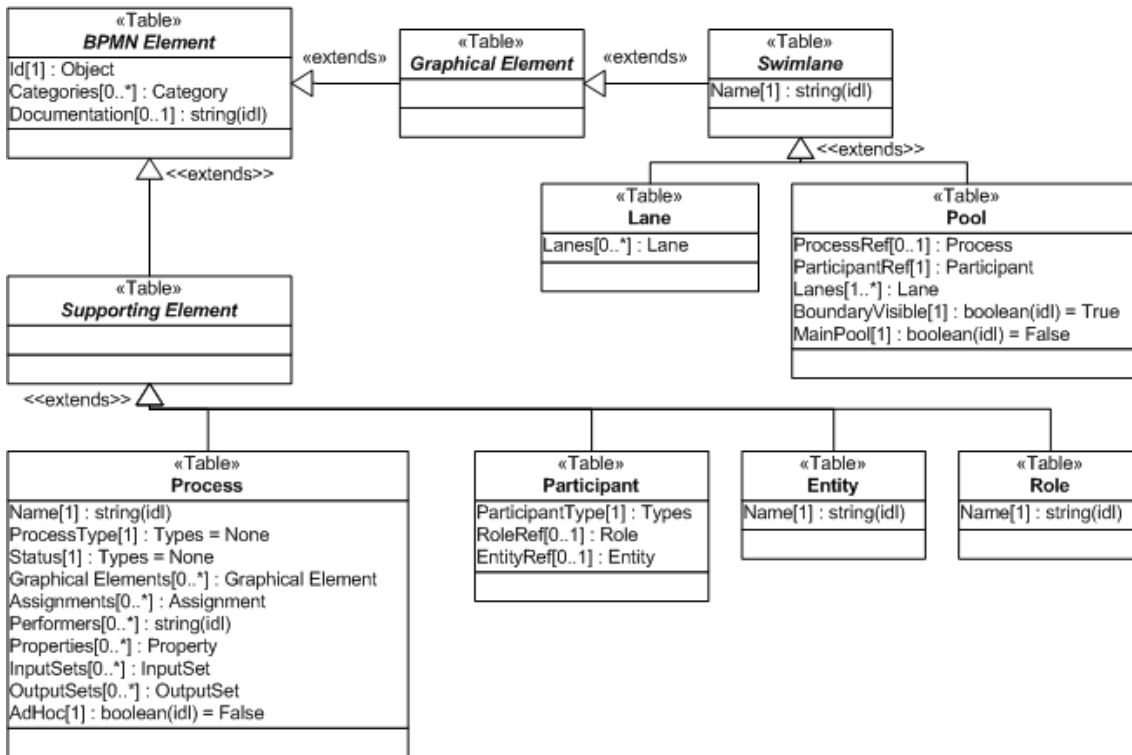
Figure B.5 - BPMN Swimlane Elements and Attributes

B.8.1 Common Swimlane Attributes

The following table displays a set of common attributes for Swimlanes (Pools and Lanes), and which extends the set of common BPMN Element attributes (see Table B.49).

Table B.75 - Common Swimlane Attributes

Attributes	Description
Name : String	Name is an attribute that is text description of the Swimlane.

B.8.2   Pool

The following table displays the identified attributes of a Pool, and which extends the set of common Swimlane attributes (see Table B.75).

Table B.76 - Pool Attributes

Attributes	Description
ProcessRef (0-1) : Process	The ProcessRef attribute defines the Process that is contained within the Pool. Each Pool MAY have a Process. The attributes for a Process can be found in "These attributes are used for Graphical Elements [which are Flow Objects (Section B.4, "Common Flow Object Attributes," on page 250), Connecting Objects (Section B.10, "Graphical Connecting Objects," on page 271), Swimlanes (Section B.8, "Swimlanes (Pools and Lanes)," on page 267), and Artifacts (Section B.9, "Artifacts," on page 269)], and Supporting Elements (Section B.11, "Supporting Elements," on page 274). on page 247."
ParticipantRef : Participant	The Modeler MUST define the Participant for a Pool. The Participant can be either a Role or an Entity. This defines the role that a particular Entity or Role the Pool will play in a Diagram that includes collaboration. The attributes for a Participant can be found in "Participant on page 284."
Lanes (1-n) : Lane	There MUST one or more Lanes within a Pool. If there is only one Lane, then that Lane shares the name of the Pool and only the Pool name is displayed. If there is more than one Lane, then each Lane has to have its own name and all names are displayed. The attributes for a Lane can be found in "Lane on page 91."
BoundaryVisible True : Boolean	This attribute defines if the rectangular boundary for the Pool is visible. Only one Pool in the Diagram MAY have the attribute set to False.
MainPool False : Boolean	This attribute defines if the Pool is the "main" Pool or the focus of the diagram. Only one Pool in the Diagram MAY have the attribute set to True.

B.8.3   Lane

The following table displays the identified attributes of a Lane, and which extends the set of common Swimlane attributes (see Table B.75).

Table B.77 - Lane Attributes

Attributes	Description
Lanes (0-*) : Lane	This attribute identifies any Lanes that are nested within the current Lane.

B.9   Artifacts

The following figure displays a diagram of the relationship between BPMN Artifact elements and their attributes (see Figure B.6).
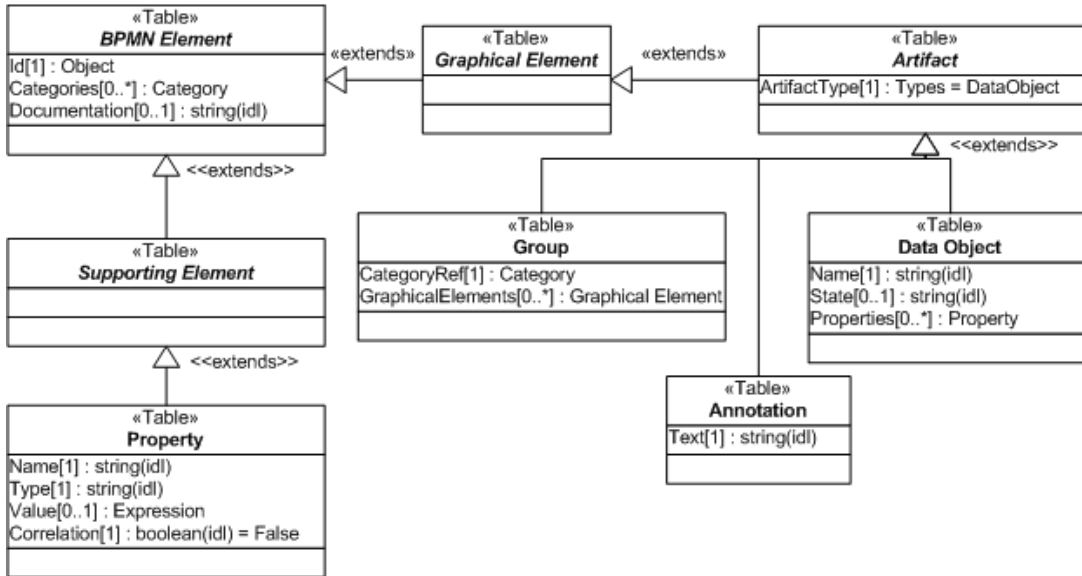
Figure B.6 - BPMN Artifact Elements and Attributes

B.9.1   Common Artifact Attributes

The following table displays the identified attributes common to Artifacts, and which extends the set of common BPMN Element attributes (see Table B.49).

Table B.78 - Common Artifact Attributes

Attributes	Description
ArtifactType (DataObject | Group | Annotation) DataObject : String	The ArtifactType MAY be set to DataObject, Group, or Annotation. The ArtifactType list MAY be extended to include new types.

B.9.2 Data Object

The following table displays the attributes for Data Objects, and which extends the set of common Artifact attributes (see Table B.78). These attributes only apply if the ArtifactType attribute is set to DataObject:

Table B.79 - Data Object Attributes

Attributes	Description
Name : String	Name is an attribute that is text description of the object.
State (0-1) : String	State is an optional attribute that indicates the impact the Process has had on the Data Object. Multiple Data Objects with the same name MAY share the same state within one Process.
Properties (0-n) : Properties	Modeler-defined Properties MAY be added to a Data Object. The fully delineated name of these properties are "<process name>.<task name>.<property name>" (e.g., "Add Customer.Review Credit Report.Score"). Further details about the definition of a Property can be found in "Property on page 285."

B.9.3 Text Annotation

The following table displays the attributes for Annotations, and which extends the set of common Artifact attributes (see Table B.78). These attributes only apply if the ArtifactType attribute is set to Annotation.

Table B.80 - Text Annotation Attributes

Attributes	Description
Text : String	Text is an attribute that is text that the modeler wishes to communicate to the reader of the Diagram.

B.9.4 Group

The following table displays the attributes for Groups, and which extends the set of common Artifact attributes (see Table B.78). These attributes only apply if the ArtifactType attribute is set to Group.

Table B.81 - Group Attributes

Attributes	Description
CategoryRef : Category	CategoryRef specifies the Category that the Group represents (Further details about the definition of a Category can be found in "Category on page 277."). The name of the Category provides the label for the Group. The graphical elements within the boundaries of the Group will be assigned the Category.
GraphicalElements (0-n) : Graphical Element	The GraphicalElements attribute identifies all of the graphical elements (e.g., Events, Activities, Gateways, and Artifacts) that are within the boundaries of the Group.

B.10 Graphical Connecting Objects

The following figure displays a diagram of the relationship between BPMN Connecting Object elements and their attributes (see Figure B.7).
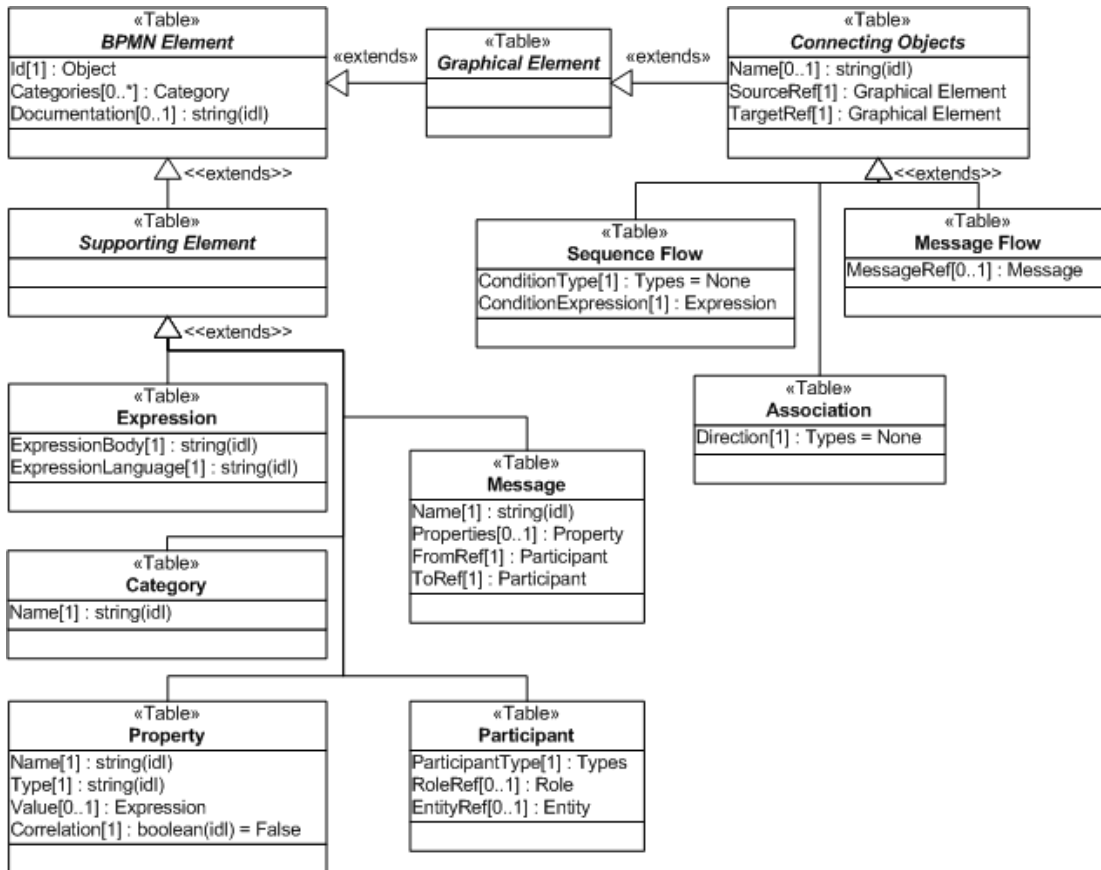
Figure B.7 - BPMN Connecting Object Elements and Attributes

B.10.1 Common Connecting Object Attributes

The following table displays the set of attributes common to Connecting Objects (Sequence Flow, Message Flow, and Association), and which extends the set of common BPMN Element attributes (see Table B.49).

Table B.82 - Common Connecting Object Attributes

Attributes	Description
Name : String	Name is an attribute that is text description of the object.
SourceRef : Graphical Element	SourceRef is an attribute that identifies which Graphical Element the Connecting Object is connected *from*. Note: there are restrictions as to what objects Sequence Flow and Message Flow can connect. Refer to the Sequence Flow Connections section and the Message Flow Connections section for each Flow Object, Swimlane, and Artifact.
TargetRef : Graphical Element	Target is an attribute that identifies which Graphical Element the Connecting Object is connected *to*. Note: there are restrictions as to what objects Sequence Flow and Message Flow can connect. Refer to the Sequence Flow Connections section and the Message Flow Connections section for each Flow Object, Swimlane, and Artifact.

B.10.2 Sequence Flow

The following table displays the set of attributes of a Sequence Flow, and which extends the set of common Connecting Object attributes (see Table B.82).

Table B.83 - Sequence Flow Attributes

Attributes	Description
ConditionType (None | Expression | Default) None : String	By default, the ConditionType of a Sequence Flow is None. This means that there is no evaluation at runtime to determine whether or not the Sequence Flow will be used. Once a Token is ready to traverse the Sequence Flow (i.e., the Source is an activity that has completed), then the Token will do so. The normal, uncontrolled use of Sequence Flow, in a sequence of activities, will have a None ConditionType (see Figure 10.1). A None ConditionType MUST NOT be used if the Source of the Sequence Flow is an Exclusive Data-Based or Inclusive Gateway. The ConditionType attribute MAY be set to Expression if the Source of the Sequence Flow is a Task, a Sub-Process, or a Gateway of type Exclusive-Data-Based or Inclusive. If the ConditionType attribute is set to Expression, then a condition marker SHALL be added to the line if the Sequence Flow is outgoing from an activity (see Figure 10.2). However, a condition indicator MUST NOT be added to the line if the Sequence Flow is outgoing from a Gateway. An Expression ConditionType MUST NOT be used if the Source of the Sequence Flow is an Event-Based Exclusive Gateway, a Complex Gateway, a Parallel Gateway, a Start Event, or an Intermediate Event. In addition, an Expression ConditionType MUST NOT be used if the Sequence Flow is associated with the Default Gate of a Gateway. The ConditionType attribute MAY be set to Default only if the Source of the Sequence Flow is an activity or an Exclusive Data-Based Gateway. If the ConditionType is Default, then the Default marker SHALL be displayed (see Figure 10.3).
[ConditionType is set to Expression only] **ConditionExpression** : Expression	If the ConditionType attribute is set to Expression, then the ConditionExpression attribute MUST be defined as a valid expression. The expression will be evaluated at runtime. If the result of the evaluation is TRUE, then a Token will be generated and will traverse the Sequence--Subject to any constraints imposed by a Source that is a Gateway.

B.10.3 Message Flow

The following table displays the identified attributes of a Message Flow, and which extends the set of common Connecting Object attributes (see Table B.82).

Table B.84 - Message Flow Attributes

Attributes	Description
MessageRef (0-1) : Message	MessageRef is an optional attribute that identifies the Message that is being sent. The attributes of a Message can be found in "Message on page 283."

B.10.4 Association

The following table displays the identified attributes of a Association, and which extends the set of common Connecting Object attributes (see Table B.82).

Table B.85 - Association Attributes

Attributes	Description
Direction (None | One | Both) None : String	Direction is an attribute that defines whether or not the Association shows any directionality with an arrowhead. The default is None (no arrowhead). A value of One means that the arrowhead SHALL be at the Target Object. A value of Both means that there SHALL be an arrowhead at both ends of the Association line.

B.11 Supporting Elements

Supporting Element is one of two main elements that are of type BPMN Element (see Figure B.1). The other is Graphical Element. There are 16 types, and a few subtypes, of Support Element. These are: These are: Assignments (see Section B.11.3, "Assignment," on page 277), Categories (see Section B.11.4, "Category," on page 277), Entities (see Section B.11.5, "Condition," on page 278), Event Details (see Section B.11.7, "Event Details," on page 278), Expressions (see Section B.11.8, "Expression," on page 281), Gates (see Section B.11.9, "Gate," on page 282), Inputs (see Section B.11.10, "InputSet," on page 282), Messages (see Section B.11.11, "Message," on page 283), Outputs (see Section B.11.13, "OutputSet," on page 284), Participants (see Section B.11.14, "Participant," on page 284), Processes (see Section B.3, "Process Attributes," on page 248), Properties (see Section B.11.15, "Property," on page 285), Roles (see Section B.11.16, "Role," on page 285), Conditions (see Section B.11.5, "Condition," on page 278), Transactions (see Section B.11.19, "Transaction," on page 286), and Web Services (see Section B.11.20, "Web Service," on page 286).

The following figure displays a diagram of the relationship between BPMN Supporting elements and their attributes (see Figure B.8).
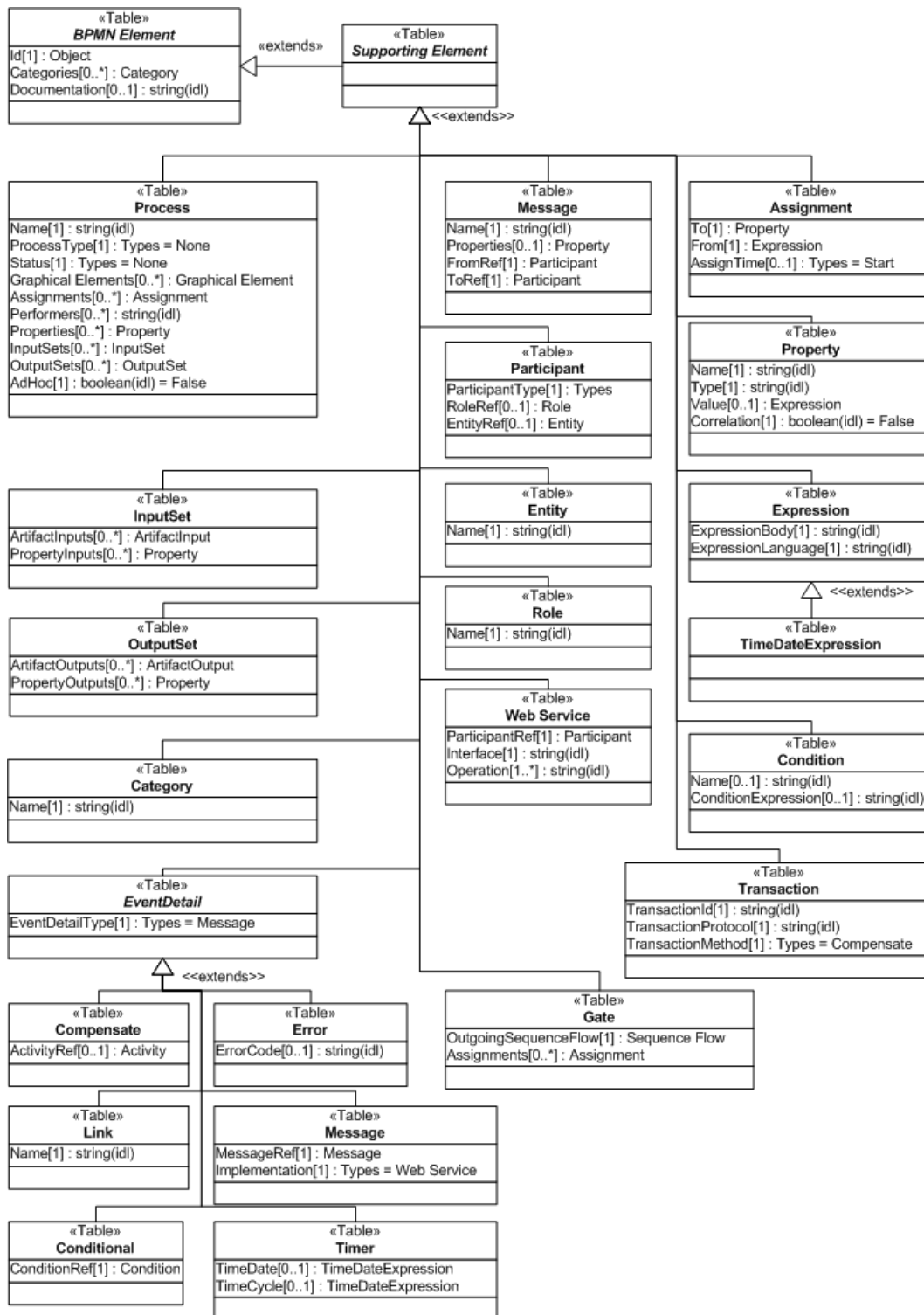
**Figure B.8 - BPMN Supporting Elements and Attributes**

B.11.1 ArtifactInput

The following table displays the set of attributes of an ArtifactInput, which is used in the definition of attributes for InputSet, and which extends the set of common BPMN Element attributes (see Table B.49).

Table B.86 - ArtifactInput Attributes

Attributes	Description
ArtifactRef : Artifact	This attribute identifies an Artifact that will be used as an input to an activity. The identified Artifact will be part of an InputSet for an activity.
RequiredForStart True : Boolean	The default value for this attribute is True. This means that the Input is required for an activity to start. If set to False, then the activity MAY start within the input if it is available, but MAY accept the input (more than once) after the activity has started. An InputSet may have a some of ArtifactInputs that have this attribute set to True and some that are set to False.

B.11.2 ArtifactOutput

The following table displays the set of attributes of an ArtifactOutput, which is used in the definition of attributes for OutputSet, and which extends the set of common BPMN Element attributes (see Table B.49).

Table B.87 - ArtifactOutput Attributes

Attributes	Description
ArtifactRef : Artifact	This attribute identifies an Artifact that will be used as an output from an activity. The identified Artifact will be part of an OutputSet for an activity.
ProduceAtCompletion True : Boolean	The default value for this attribute is True. This means that the Output will be produced when an activity has been completed. If set to False, then the activity MAY produce the output (more than once) before it has completed. An OutputSet may have a some of ArtifactOutputs that have this attribute set to True and some that are set to False.

B.11.3 Assignment

The following table displays the set of attributes of an Assignment, which is used in the definition of attributes for Process, Activities, Events, Gateways, and Gates, and which extends the set of common BPMN Element attributes (see Table B.49):

Table B.88 - Assignment Attributes

Attributes	Description
To : Property	The target for the Assignment MUST be a Property of the Process or the activity itself.
From : Expression	The Expression MUST be made up of a combination of Values, Properties, and Attributes, which are separated by operators such as add or multiply. The expression language is defined in the ExpressionLanguage attribute of the Business Process Diagram - see "Business Process Diagram Attributes on page 247."
AssignTime (0-1) (Start | End) Start : String	An Assignment MAY have a AssignTime setting. If the Object is an activity (Task, Sub-Process, or Process), then the Assignment MUST have an AssignTime. A value of Start means that the assignment SHALL occur at the start of the activity. This can be used to assign the higher-level (global) Properties of the Process to the (local) Properties of the activity as an input to the activity. A value of End means that the assignment SHALL occur at the end of the activity. This can be used to assign the (local) Properties of the activity to the higher-level (global) Properties of the Process as an output to the activity.

B.11.4 Category

The following table displays the set of attributes of a Category, which is used in the definition of attributes for all BPMN elements, and which extends the set of common BPMN Element attributes (see Table B.49). Since a Category is also a BPMN element, a Category can have Categories to create a hierarchical structure of Categories.

Table B.89 - Category Attributes

Attributes	Description
Name : String	Name is an attribute that is text description of the Category and is used to visually distinguish the category.

B.11.5 Condition

The following table displays the set of attributes of a Condition, which is used in the definition of attributes for Start Event and Intermediate Event, and which extends the set of common BPMN Element attributes (see Table B.49):

Table B.90 - Condition Attributes

Attributes	Description
Name (0-1): String	Name is an optional attribute that is text description of the Condition. If a Name is not entered, then a ConditionExpression MUST be entered (see the attribute below).:
ConditionExpression: (0-1) : Expression	A ConditionExpression May be entered. In some cases the Condition itself will be stored and maintained in a separate application (e.g., a Rules Engine). If a ConditionExpression is not entered, then a Name MUST be entered (see the attribute above). The attributes of an Expression can be found in "Expression on page 281."

B.11.6 Entity

The following table displays the set of attributes of an Entity, which is used in the definition of attributes for a Participant, and which extends the set of common BPMN Element attributes (see Table B.49):

Table B.91 - Entity Attributes

Attributes	Description
Name : String	Name is an attribute that is text description of the Entity.

B.11.7 Event Details

The following sections will present the attributes common to all Event Details and the specific attributes for the Event Details that have additional attributes. Note that the Cancel and Terminate Event Details do not have additional attributes.

Common EventDetail Attributes

The following table displays the set of attributes common to the types of EventDetail, and which extends the set of common BPMN Element attributes (see Table B.49).

Table B.92 - Common EventDetail Attributes

Attributes	Description
EventDetailType (Message | Timer | Error | Conditional | Link | Signal | Compensate | Cancel | Terminate) Message : String	The EventDetailType attribute defines the type of trigger expected for an Event. The set of types includes Message, Timer, Error, Conditional, Link, Signal, Compensate, Cancel, and Terminate. The EventTypes (Start, Intermediate, and End) will each have a subset of the EventDetailTypes that can be used. The EventDetailType list MAY be extended to include new types. These new types MAY have a new modeler- or tool-defined Marker to fit within the boundaries of the Event.

Conditional Event Detail

The following table displays the set of attributes a Conditional EventDetail, and which extends the set of common Event Detail attributes (see Table B.92).

Table B.93 - Conditional EventDetail Attributes

Attributes	Description
ConditionRef : Condition	If the Trigger is Conditional, then a Condition MUST be entered. The attributes of a Condition can be found in Section B.11.5, "Condition," on page 278.

Compensation Event Detail

The following table displays the set of attributes a Compensation EventDetail, and which extends the set of common Event Detail attributes (see Table B.92).

Table B.94 - Compensation EventDetail Attributes

Attributes	Description
ActivityRef (0-1): Activity	*For an End Event:* If the Result is a Compensation, then the Activity that needs to be compensated MAY be supplied. If an Activity is not supplied, then the Event broadcast to all completed activities in the Process Instance. *For an Intermediate Event within Normal Flow:* If the Trigger is a Compensation, then the Activity that needs to be compensated MAY be supplied. If an Activity is not supplied, then the Event broadcast to all completed activities in the Process Instance. This "throws" the compensation. *For an Intermediate Event attached to the boundary of an Activity:* This Event "catches" the compensation. No further information is required. The Activity the Event is attached to will provide the Id necessary to match the compensation event with the event that "threw" the compensation or the compensation will be a broadcast.

Error Event Detail

The following table displays the set of attributes a Error EventDetail, and which extends the set of common Event Detail attributes (see Table B.92).

Table B.95 - Error EventDetail Attributes

Attributes	Description
ErrorCode : String	*For an End Event:* If the Result is an Error, then the ErrorCode MUST be supplied.This "throws" the error. *For an Intermediate Event within Normal Flow:* If the Trigger is an Error, then the ErrorCode MUST be entered. This "throws" the error. *For an Intermediate Event attached to the boundary of an Activity:* If the Trigger is an Error, then the ErrorCode MAY be entered. This Event "catches" the error. If there is no ErrorCode, then any error SHALL trigger the Event. If there is an ErrorCode, then only an error that matches the ErrorCode SHALL trigger the Event.

Link Event Detail

The following table displays the set of attributes a Link EventDetail, and which extends the set of common Event Detail attributes (see Table B.92).

Table B.96 - Link EventDetail Attributes

Attributes	Description
Name : String	If the Trigger is a Link, then the Name MUST be entered.

Message Event Detail

The following table displays the set of attributes a Message EventDetail, and which extends the set of common Event Detail attributes (see Table B.92).

Table B.97 - Message EventDetail Attributes

Attributes	Description
MessageRef : Message	If the EventDetailType is a MessageRef, then the a Message MUST be supplied. The attributes of a Message can be found in Section B.11.11, "Message," on page 283.
Implementation (Web Service | Other | Unspecified) Web Service	This attribute specifies the technology that will be used to send or receive the message. A Web service is the default technology.

Signal Event Detail

The following table displays the set of attributes a Signal EventDetail, and which extends the set of common Event Detail attributes (see Table B.92).

Table B.98 - Signal EventDetail Attributes

Attributes	Description
SignalRef : String	If the Trigger is a Signal, then a Signal Shall be entered. The attributes of a Signal can be found in Section B.11.17, "Signal," on page 285.

Timer Event Detail

The following table displays the set of attributes a Timer EventDetail, and which extends the set of common Event Detail attributes (see Table B.92).

Table B.99 - Timer EventDetail Attributes

Attributes	Description
TimeDate (0-1) : TimeDateExpression	If the Trigger is a Timer, then a TimeDate MAY be entered. If a TimeDate is not entered, then a TimeCycle MUST be entered (see the attribute below). The attributes of a TimeDateExpression can be found in Section B.11.18, "TimeDateExpression," on page 285
TimeCycle (0-1) : TimeDateExpression	If the Trigger is a Timer, then a TimeCycle MAY be entered. If a TimeCycle is not entered, then a TimeDate MUST be entered (see the attribute above).

B.11.8 Expression

The following table displays the set of attributes of an Expression, which is used in the definition of attributes for Start Event, Intermediate Event, Activity, Complex Gateway, and Sequence Flow, and which extends the set of common BPMN Element attributes (see Table B.49):

Table B.100 - Expression Attributes

Attributes	Description
ExpressionBody : String	An ExpressionBody MUST be entered to provide the text of the expression, which will be written in the language defined by the ExpressionLanguage attribute.
ExpressionLanguage : String	A Language MUST be provided to identify the language of the ExpressionBody. The value of the ExpressionLanguage should follow the naming conventions for the version of the specified language.

B.11.9 Gate

The following table displays the set of attributes of a Gate, which is used in the definition of attributes for Gateways, and which extends the set of common BPMN Element attributes (see Table B.49).

Table B.101 - Gate Attributes

Attributes	Description
OutgoingSequenceFlowRef : Sequence Flow	Each Gate MUST have an associated (outgoing) Sequence Flow. The attributes of a Sequence Flow can be found in the Section B.10.2, "Sequence Flow," on page 273. *For Exclusive Event-Based, Complex, and Parallel Gateways:* The Sequence Flow MUST have its Condition attribute set to None (there is not an evaluation of a condition expression). *For Exclusive Data-Based, and Inclusive Gateways:* The Sequence Flow MUST have its Condition attribute set to Expression and MUST have a valid ConditionExpression. The ConditionExpression MUST be unique for all the Gates within the Gateway. If there is only one Gate (i.e., the Gateway is acting only as a Merge), then Sequence Flow MUST have its Condition set to None. *For DefaultGates:* The Sequence Flow MUST have its Condition attribute set to Otherwise
Assignments (0-n) : Assignment	One or more assignment expressions MAY be made for each Gate. The Assignment SHALL be performed when the Gate is selected. The Assignment is defined in the Section B.11.3, "Assignment," on page 277.

B.11.10 InputSet

The following table displays the set of attributes of an InputSet, which is used in the definition of common attributes for Activities and for attributes of a Process, and which extends the set of common BPMN Element attributes (see Table B.49):

Table B.102 - Input Attributes

Attributes	Description
ArtifactInputs (0-n) : ArtifactInput	Zero or more ArtifactInputs MAY be defined for each InputSet. For the combination of ArtifactInputs and PropertyInputs, there MUST be at least one item defined for the InputSet. An ArtifactInput is an Artifact, usually a Data Object. Note that the Artifacts MAY also be displayed on the diagram and MAY be connected to the activity through an Association--however, it is not required for them to be displayed. Further details about the definition of an ArtifactInput can be found in Section B.11.1, "ArtifactInput," on page 276.
PropertyInputs (0-n) : Property	Zero or more PropertyInputs MAY be defined for each InputSet. For the combination of ArtifactInputs and PropertyInputs, there MUST be at least one item defined for the InputSet.

B.11.11Message

The following table displays the set of attributes of a Message, which is used in the definition of attributes for a Start Event, End Event, Intermediate Event, Task, and Message Flow, and which extends the set of common BPMN Element attributes (see Table B.49):

Table B.103 - Message Attributes

Attributes	Description
Name : String	Name is an attribute that is text description of the Message.
Properties (0-n) : Property	Multiple Properties MAY entered for the Message. The attributes of a Property can be found in "Property on page 285."
FromRef : Participant	This defines the source of the Message. The attributes for a Participant can be found in "Participant on page 284."
ToRef : Participant	This defines the target of the Message. The attributes for a Participant can be found in "Participant on page 284."

B.11.12Object

The following table displays the set of attributes of an Object, which is used in the definition of attributes for all graphical elements.

Table B.104 - Object Attributes

Attributes	Description
Id : String	The Id attribute provides a unique identifier for all objects on a diagram. That is, each object MUST have a different value for the ObjectId attribute.

B.11.13OutputSet

The following table displays the set of attributes of an OutputSet, which is used in the definition of common attributes for Activities and for attributes of a Process, and which extends the set of common BPMN Element attributes (see Table B.49):

Table B.105 - Output Attributes

Attributes	Description
ArtifactOutputs (0-n) : Arti-factOutput	Zero or more ArtifactOutputs MAY be defined for each InputSet. For the combination of ArtifactOutputs and PropertyOutputs, there MUST be at least one item defined for the OutputSet. An ArtifactOutput is an Artifact, usually a Data Object. Note that the Artifacts MAY also be displayed on the diagram and MAY be connected to the activity through an Association--however, it is not required for them to be displayed. Further details about the definition of an ArtifactOutput can be found in Section B.11.2, "ArtifactOutput," on page 276.
PropertyOutputs (0-n) : Property	Zero or more PropertyOutputs MAY be defined for each InputSet. For the combination of ArtifactOutputs and PropertyOutputs, there MUST be at least one item defined for the OutputSet.

B.11.14Participant

The following table displays the set of attributes of a Participant, which is used in the definition of attributes for a Pool, Message, and Web service, and which extends the set of common BPMN Element attributes (see Table B.49):

Table B.106 - Participant Attributes

Attributes	Description
ParticipantType (Role | Entity) Role : String	Each Property has a Name (e.g., name="Customer Name").
[ParticipantType = "Role" only] **RoleRef** (0-1) : Role	If the ParticipantType = Role, then a Role MUST be identified. The attributes for a Role can be found in "Role on page 285."
[ParticipantType = "Entity" only] **EntityRef** (0-1) : Entity	If the ParticipantType = Entity, then an Entity MUST be identified. The attributes for an Entity can be found in "Condition on page 278."

B.11.15Property

The following table displays the set of attributes of a Property, which is used in the definition of attributes for a Process and common activity attributes, and which extends the set of common BPMN Element attributes (see Table B.49):

Table B.107 - Property Attributes

Attributes	Description
Name : String	Each Property has a Name (e.g., name="Customer Name").
Type : String	Each Property has a Type (e.g., type="String"). Properties may be defined hierarchically.
Value (0-1) : Expression	Each Property MAY have a Value specified.
Correlation (0-1) False : Boolean	If the Correlation attribute is set to True, then the Property is marked to be used for correlation (e.g., for incoming Messages).

B.11.16Role

The following table displays the set of attributes of a Role, which is used in the definition of attributes for a Participant, and which extends the set of common BPMN Element attributes (see Table B.49):

Table B.108 - Role Attributes

Attributes	Description
Name : String	Name is an attribute that is text description of the Role.

B.11.17Signal

The following table displays the set of attributes of a Signal, which is used in the definition of attributes for a Start Event, End Event, Intermediate Event, and which extends the set of common BPMN Element attributes (see Table B.49).

Table B.109 - Message Attributes

Attributes	Description
Name : String	Name is an attribute that is text description of the Signal.
Properties (0-n) : Property	Multiple Properties MAY entered for the Signal. The attributes of a Property can be found in "Property on page 285."

B.11.18TimeDateExpression

The TimeDateExpression supporting element is a sub-type of the Expression Element (Expression on page 281) and uses all the attributes of the Expression Element.

B.11.19Transaction

The following table displays the set of attributes of a Transaction, which is used in the definition of attributes for a Sub-Process, and which extends the set of common BPMN Element attributes (see Table B.49):

Table B.110 - Transaction Attributes

Attributes	Description
TransactionId : String	The TransactionId attribute provides an identifier for the Transactions used within a diagram.
TransactionProtocol : String	This identifies the Protocol (e.g., WS-Transaction or BTP) that will be used to control the transactional behavior of the Sub-Process.
TransactionMethod (Compensate | Store | Image) Compensate : String	TransactionMethod is an attribute that defines the technique that will be used to undo a Transaction that has been cancelled. The default is Compensate, but the attribute MAY be set to Store or Image.

B.11.20Web Service

The following table displays the set of attributes of an Web Service, which is used in the definition of attributes for Message Start Event, Message Intermediate Event, Message End Event, Receive Task, Send Task, Service Task, and User Task, and which extends the set of common BPMN Element attributes (see Table B.49):

Table B.111 - Web Service Attributes

Attributes	Description
ParticipantRef : Participant	A Participant for the Web Service MUST be entered. The attributes for a Participant can be found in "Participant on page 284."
Interface : String	(aka portType) An Interface for the Web Service MUST be entered.
Operation (1-n): String	One or more Operations for the Web Service MUST be entered.

<div align="center">

Annex C
(informative)

Glossary

</div>

A

Activity: An activity is a generic term for work that company or organization performs via business processes. An activity can be atomic or non-atomic (compound). The types of activities that are a part of a Process Model are: Process, Sub-Process, and Task.

Abstract Process: An Abstract Process represents the interactions between a private business process and another process or participant.

AND-Join: (from the WfMC Glossary[1]) An AND-Join is a point in the <u>Process</u> where two or more parallel executing activities converge into a single common thread of <u>Sequence Flow</u>. See "Join."

AND-Split: (from the WfMC Glossary[2]) An AND-Split is a point in the <u>Process</u> where a single thread of <u>Sequence Flow</u> splits into two or more threads which are executed in parallel within the <u>Process</u>, allowing multiple activities to be executed simultaneously. See "Fork."

Arbitrary Cycles: (From the Workflow Patterns Initiative[2]). Pattern #11: A point in a workflow process when one or more activities can be done repeatedly[3].

Artifact: An Artifact is a graphical object that provides supporting information about the Process or elements within the Process. However, it does not directly affect the flow of the Process. BPMN has standardized the shape of a Data Object. Other examples of Artifacts include critical success factors and milestones.

---

1. The underlined terms in this definition were changed from the original definition. "Process" is used in place of "workflow." "Sequence Flow" is used in place of "control."
2. http://tmitwww.tm.tue.nl/research/patterns/patterns.htm
3. http://tmitwww.tm.tue.nl/research/patterns/arbitrary_cycles.htm

Association:	An Association is a dotted graphical line that is used to associate information and Artifacts with Flow Objects. Text and graphical non-Flow Objects can be associated with the Flow Objects and Flow.
Atomic Activity:	An atomic activity is an activity not broken down to a finer level of Process Model detail. It is a leaf in the tree-structure hierarchy of Process activities. Graphically it will appear as a Task in BPMN.

B

Business Analyst:	A Business Analyst is an individual within an organization who defines, manages, or monitors Business Processes. They are usually distinguished from the IT specialists or programmers who implement the Business Process within a BPMS.
Business Process:	A Business Process is displayed within a Business Process Diagram (BPD). A Business Process contains one or more Processes.
Business Process Diagram:	A Business Process Diagram (BPD) is the diagram that is specified by BPMN. A BPD uses the graphical elements and that semantics that support these elements as defined in this specification.
Business Process Management:	Business Process Management (BPM) encompasses the discovery, design, and deployment of business processes. In addition, BPM includes the executive, administrative, and supervisory control of those processes[4].
BPM System:	The technology that enables BPM.

C

Cancel Activity:	(From the Workflow Patterns Initiative[5]). Pattern #20: An enabled activity is disabled, i.e. a thread waiting for the execution of an activity is removed[6].
Cancel Case:	(From the Workflow Patterns Initiative[2]). Pattern #21: A case, i.e.workflow instance, is removed completely[7].
Choreography:	Choreography is an ordered sequence of B2B message exchanges.

---

4. From "Business Process Management: the Third Wave," by Howard Smith and Peter Fingar, pg 4. 2003, Meghan-Kiffer Press. ISBN 0-929652-33-9
5. http://tmitwww.tm.tue.nl/research/patterns/patterns.htm
6. http://tmitwww.tm.tue.nl/research/patterns/cancel_activity.htm
7. http://tmitwww.tm.tue.nl/research/patterns/cancel_case.htm

Collaboration:	Collaboration is the act of sending messages between any two Participants in a BPMN model. The two Participants represent two separate BPML processes.
Collaboration Process:	A Collaboration Process depicts the interactions between two or more business entities.
Collapsed Sub-Process:	A Collapsed Sub-Process is a Sub-Process that hides its flow details. The Collapsed Sub-Process object uses a marker to distinguish it as a Sub-Process, rather than a Task. The marker is a small square with a plus sign (+) inside.
Compensation Flow:	Compensation Flow is defines the set of activities that are performed during the roll-back of a transaction to compensate for activities that were performed during the Normal Flow of the Process. Compensation can also be called from a Compensate End or Intermediate Event.
Compound Activity:	A compound activity is an activity that has detail that is defined as a flow of other activities. It is a branch (or trunk) in the tree-structure hierarchy of Process activities. Graphically, it will appear as a Process or Sub-Process in BPMN.
Controlled Flow:	Flow that proceeds from one Flow Object to another, via a Sequence Flow link, but is subject to either conditions or dependencies from other flow as defined by a Gateway. Typically, this is seen as a Sequence flow between two activities, with a conditional indicator (mini-diamond) or a Sequence Flow connected to a Gateway.

D

Decision:	Decisions are locations within a business process where the Sequence Flow can take two or more alternative paths. This is basically the "fork in the road" for a process. For a given performance (or instance) of the process, only one of the forks can be taken. A Decision is a type of Gateway. See "Or-Split."

Deferred Choice:	(From the Workflow Patterns Initiative[8]). Pattern #17: A point in the workflow process where one of several branches is chosen. In contrast to the exclusive split, the choice is not made explicitly (e.g. based on data or a decision) but several alternatives are offered to the environment. However, in contrast to the fork, only one of the alternatives is executed. This means that once the environment activates one of the branches the other alternative branches are withdrawn. It is important to note that the choice is delayed until the processing in one of the alternative branches is actually started, i.e. the moment of choice is as late as possible[9].
Discriminator:	(From the Workflow Patterns Initiative[1]). Pattern #8: The discriminator is a point in a workflow process that waits for a number of incoming branches to complete before activating the subsequent activity. From that moment on it waits for all remaining branches to complete and "ignores" them. Once all incoming branches have been triggered, it resets itself so that it can be triggered again[10].

E

End Event:	As the name implies, the End Event indicates where a process will end. In terms of Sequence Flow, the End Event ends the flow of the Process, and thus, will not have any outgoing Sequence Flow. An End Event can have a specific Result that will appear as a marker within the center of the End Event shape. End Event Results are Message, Error, Compensation, Link, and Multiple. The End Event shares the same basic shape of the Start Event and Intermediate Event, a circle, but is drawn with a thick single line
Event Context:	An Event Context is the set of activities that can be interrupted by an exception (Intermediate Event). This can be one activity or a group of activities in an expanded Sub-Process.
Exception:	An Exception is an event that occurs during the performance of the process that causes Normal Flow of the process to be diverted exclusively from Normal Flow. Exceptions can be generated by a time out, fault, message, etc.

---

8. http://tmitwww.tm.tue.nl/research/patterns/patterns.htm
9. http://tmitwww.tm.tue.nl/research/patterns/deferred_choice.htm
10. http://tmitwww.tm.tue.nl/research/patterns/discriminator.htm

Exception Flow:	Exception Flow is a set of Sequence Flow that originates from an Intermediate Event that is attached to the boundary of an activity. The Process will not traverse this flow unless an Exception occurs during the performance of that activity (through an Intermediate Event).
Exclusive Choice:	(From the Workflow Patterns Initiative[11]). Pattern #4: A point in the workflow process where, based on a decision or workflow control data, one of several branches is chosen[12].
Expanded Sub-Process:	An Expanded Sub-Process is a Sub-Process that exposes its flow detail within the context of its Parent Process. It will maintain its rounded rectangle shape, but will be enlarged to a size sufficient to display the Flow Objects within.

F

Flow:	A Flow is a graphical line connecting two objects in a BPD. There are two types of Flow: Sequence Flow and Message Flow, each with their own line style. Flow is also used in a generic sense (and lowercase) to describe how Tokens will traverse Sequence Flow from the Start Event to an End Event.
Flow Object:	A Flow Object is one of the set of following graphical objects: Events, Activities, and Gateways.
Fork:	A fork is a point in the Process where a single flow is divided into two or more Flow. It is a mechanism that will allow activities to be performed concurrently, rather than sequentially. BPMN uses multiple outgoing Sequence Flow or an Parallel Gateway to perform a Fork. See "AND-Split."

I

Implicit Termination:	(From the Workflow Patterns Initiative[13]). Pattern #12: A given subprocess should be terminated when there is nothing else to be done. In other words, there are no active activities in the workflow and no other activity can be made active (and at the same time the workflow is not in deadlock)[14].

---

11. http://tmitwww.tm.tue.nl/research/patterns/patterns.htm
12. http://tmitwww.tm.tue.nl/research/patterns/exclusive_choice.htm
13. http://tmitwww.tm.tue.nl/research/patterns/patterns.htm
14. http://tmitwww.tm.tue.nl/research/patterns/implicit_termination.htm

Interleaved Parallel Routing:	(From the Workflow Patterns Initiative[1]). Pattern #18: A set of activities is executed in an arbitrary order: Each activity in the set is executed, the order is decided at run-time, and no two activities are executed at the same moment (i.e.no two activities are active for the same workflow instance at the same time)[15].
Intermediate Event:	An Intermediate Event is an event that occurs after a Process has been started. It will affect the flow of the process, but will not start or (directly) terminate the process. An Intermediate Event will show where messages or delays are expected within the Process, disrupt the Normal Flow through exception handling, or show the extra flow required for compensating a transaction. The Intermediate Event shares the same basic shape of the Start Event and End Event, a circle, but is drawn with a thin double line.

J

Join:	A Join is a point in the Process where two or more parallel Sequence Flow are combined into one Sequence Flow. BPMN uses an Parallel Gateway to perform a Join. See "AND-Join."

L

Lane:	An 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. The meaning of the Lanes is up to the modeler.

M

Merge:	A Merge is a point in the process where two or more alternative Sequence Flow are combined into one Sequence Flow. BPMN uses multiple incoming Sequence Flow or an Exclusive Gateway to perform a Merge. See "OR-Join."
Message:	A Message is the object that is transmitted through a Message Flow. The Message will have an identity that can be used for alternative branching of a Process through the Event-Based Exclusive Gateway.

---

15. http://tmitwww.tm.tue.nl/research/patterns/interleaved_parallel_routing.htm

Message Flow:	A Message Flow is a dashed line that 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.
Milestone:	(From the Workflow Patterns Initiative[16]). Pattern #19: The enabling of an activity depends on the case being in a specified state, i.e.the activity is only enabled if a certain milestone has been reached which did not expire yet. Consider three activities A, B, and C. Activity A is only enabled if activity B has been executed and C has not been executed yet, i.e.A is not enabled before the execution B and A is not enabled after the execution C[17].
Multiple Choice:	(From the Workflow Patterns Initiative[1]). Pattern #6: A point in the workflow process where, based on a decision or workflow control data, one or more branches are chosen[18].
Multiple Instances:	(From the Workflow Patterns Initiative[1]). Patterns #13-16: There are four defined patterns. 1. For one case an activity is enabled multiple times. The number of instances of a given activity for a given case is known at design time. 2. For one case an activity is enabled multiple times. The number of instances of a given activity for a given case is variable and may depend on characteristics of the case or availability of resources, but is known at some stage during runtime, before the instances of that activity have to be created. 3. For one case an activity is enabled multiple times. The number of instances of a given activity for a given case is not known during design time, nor it is known at any stage during runtime, before the instances of that activity have to be created. 4 For one case an activity is enabled multiple times. The number of instances may not be known at design time. After completing all instances of that activity another activity has to be started[19].
Multiple Merge:	(From the Workflow Patterns Initiative[1]). Pattern #7: Multi-merge is a point in a workflow process where two or more branches reconverge without synchronization. If more than one branch gets activated, possibly concurrently, the activity following the merge is started **once for every incoming branch that gets activated**[20].

---

16. http://tmitwww.tm.tue.nl/research/patterns/patterns.htm
17. http://tmitwww.tm.tue.nl/research/patterns/milestone.htm
18. http://tmitwww.tm.tue.nl/research/patterns/multiple_choice.htm
19. http://tmitwww.tm.tue.nl/research/patterns/patterns.htm

N

N-out-of-M-Join: (From the Workflow Patterns Initiative[1]). Pattern #9: N-out-of-M Join is a point in a workflow process where M parallel paths converge into one. The subsequent activity should be activated once N paths have completed. Completion of all remaining paths should be ignored. Similarly to the discriminator, once all incoming branches have "fired", the join resets itself so that it can fire again[21].

Normal Flow: Normal Flow is the flow that originates from a Start Event and continues through activities via alternative and parallel paths until it ends at an End Event.

O

OR-Join: (from the WfMC Glossary[22]) An Or-Join is a point in the <u>Process</u> where two or more alternative activity(s) <u>Process</u> branches re-converge to a single common activity as the next step within the <u>Process</u>. (As no parallel activity execution has occurred at the join point, no synchronization is required.) See "Merge."

OR-Split: (from the WfMC Glossary[1]) An OR-Split is a point in the <u>Process</u> where a single thread of <u>Sequence Flow</u> makes a decision upon which branch to take when encountered with multiple alternative <u>Process</u> branches. See "Decision."

P

Parallel Split: (From the Workflow Patterns Initiative[23]). Pattern #2: Parallel split is required when two or more activities need to be executed in parallel. Parallel split is easily supported by most workflow engines except for the most basic scheduling systems that do not require any degree of concurrency[24].

Parent Process: A Parent Process is the Process that holds a Sub-Process within its boundaries.

---

20. http://tmitwww.tm.tue.nl/research/patterns/multiple_merge.htm
21. http://tmitwww.tm.tue.nl/research/patterns/n-out-of-m_join.htm
22. The underlined terms in this definition were changed from the original definition. "Process" is used in place of "workflow." "Sequence Flow" is used in place of "control."
23. http://tmitwww.tm.tue.nl/research/patterns/patterns.htm
24. http://tmitwww.tm.tue.nl/research/patterns/parallel_split.htm

Participant:	A Participant is a business entity (e.g., a company, company division, or a customer) or a business role (e.g., a buyer or a seller), which controls or is responsible for a business process. If Pools are used, then a Participant would be associated with one Pool.
Pool:	A Pool represents a Participant in a Process. It also acts as a "swimlane" and a graphical container for partitioning a set of activities from other Pools, usually in the context of B2B situations. It is a square-cornered rectangle that is drawn with a solid single line. A Pool acts as the container for the Sequence Flow between activities. The Sequence Flow can cross the boundaries between Lanes of a Pool, but cannot cross the boundaries of a Pool. The interaction between Pools, e.g., in a B2B context, is shown through Message Flow.
Private Business Process:	A private business process is internal to a specific organization and is the type of process that has been generally called a workflow or BPM process. A single private business process will map to a single BPML document.
Process:	A Process is any activity performed within a company or organization. In BPMN a Process is depicted as a network of Flow Objects, which are a set of other activities and the controls that sequence them.

R

Result:	A Result is consequence of reaching an End Event. Results can be of different types, including: Message, Error, Compensation, Link, and Multiple.

S

Sequence:	(From the Workflow Patterns Initiative[25]). Pattern #1: Sequence is the most basic workflow pattern. It is required when there is a dependency between two or more tasks so that one task cannot be started (scheduled) before another task is finished[26].
Sequence Flow:	A Sequence Flow is a solid graphical line that is used to show the order that activities will be performed in a Process. Each Flow has only one source and only one target.

---

25. http://tmitwww.tm.tue.nl/research/patterns/patterns.htm
26. http://tmitwww.tm.tue.nl/research/patterns/sequence.htm

Simple Merge:	(From the Workflow Patterns Initiative^{"http://tmitwww.tm.tue.nl/research/patterns/patterns.htm" on page 295}). Pattern #5: A point in the workflow process where two or more alternative branches come together without synchronization. In other words the merge will be triggered once any of the incoming transitions are triggered[27].

Simple Merge: (From the Workflow Patterns Initiative"http://tmitwww.tm.tue.nl/ research/patterns/patterns.htm" on page 295). Pattern #5: A point in the workflow process where two or more alternative branches come together without synchronization. In other words the merge will be triggered once any of the incoming transitions are triggered[27].

Start Event: A Start Event indicates where a particular Process will start. In terms of Sequence Flow, the Start Event starts the flow of the Process, and thus, will not have any incoming Sequence Flow. A Start Event can have a Trigger that indicates how the Process starts: Message, Timer, Rule, Link, or Multiple. The Start Event shares the same basic shape of the Intermediate Event and End Event, a circle, but is drawn with a single thin line

Sub-Process: A Sub-Process is Process that is included within another Process. The Sub-Process can be in a collapsed view that hides its details. A Sub-Process can be in an expanded view that shows its details within the view of the Process in which it is contained. A Sub-Process shares the same shape as the Task, which is a rectangle that has rounded corners.

Swimlane: A Swimlane is a graphical container for partitioning a set of activities from other activities. BPMN has two different types of Swimlanes. See "Pool" and "Lane."

Synchronizing Join: (From the Workflow Patterns Initiative[28]). Pattern #10: A point in the workflow process where multiple paths converge into one single thread. If more than one path is taken, synchronization of the active threads needs to take place. If only one path is taken, the alternative branches should reconverge without synchronization[29].

Synchronization: (From the Workflow Patterns Initiative"http://tmitwww.tm.tue.nl/ research/patterns/patterns.htm" on page 295). Pattern #3: Synchronization is required when an activity can be started only when two parallel threads complete[30].

---

27. http://tmitwww.tm.tue.nl/research/patterns/simple_merge.htm
28. http://tmitwww.tm.tue.nl/research/patterns/patterns.htm
29. http://tmitwww.tm.tue.nl/research/patterns/synchronizing_join.htm
30. http://tmitwww.tm.tue.nl/research/patterns/synchronization.htm

T

Task: 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. Generally, an end-user and/or an application are used to perform the Task when it is executed. A Task object shares the same shape as the Sub-Process, which is a rectangle that has rounded corners.

Token: A Token is a descriptive construct used to describe how the flow of a process will proceed at runtime. By tracking how the Token traverses the Flow Objects, gets diverted through alternative paths, and gets split into parallel paths, the normal Sequence Flow should be completely definable.A Token will have a unique identity that can be used to separate multiple Tokens that may exist because of concurrent process instances or the splitting of the Token for parallel processing within a single process instance.

Transaction: A Transaction is a set of coordinated activities carried out by independent, loosely-coupled systems in accordance with a contractually defined business relationship. This coordination leads to an agreed, consistent, and verifiable outcome across all participants.

Trigger: A Trigger is a mechanism that signals the start of a business process. Triggers are associated with a Start Events and Intermediate Events and can be of the type: Message, Timer, Rule, Link, and Multiple.

U

Uncontrolled Flow: Flow that proceeds, unrestricted, from one Flow Object to another, via a Sequence Flow link, without any dependencies on another flow or any conditional expressions. Typically, this is seen as a Sequence flow between two activities, without a conditional indicator (mini-diamond) or any intervening Gateway.