Date: November 2008

Business Process Definition MetaModel Volume II: Process Definitions

Version 1.0

OMG Document Number: formal/2008-11-04
Standard document URL: http://www.omg.org/spec/BPDM/1.0
Associated File(s)*: http://www.omg.org/spec/BPDM/20080501

http://www.omg.org/spec/BPDM/20080501/Abstractions.xsd
http://www.omg.org/spec/BPDM/20080501/Activity.xsd
http://www.omg.org/spec/BPDM/20080501/BehaviorModel.xsd
http://www.omg.org/spec/BPDM/20080501/bpdm.xsd
http://www.omg.org/spec/BPDM/20080501/bpmn.cmof
http://www.omg.org/spec/BPDM/20080501/BPMNLibrary
http://www.omg.org/spec/BPDM/20080501/CommonInfrastructure.cmof
http://www.omg.org/spec/BPDM/20080501/CommonInfrastructureLibrary
http://www.omg.org/spec/BPDM/20080501/CompositionModel.xsd
http://www.omg.org/spec/BPDM/20080501/ConditionModel.xsd
http://www.omg.org/spec/BPDM/20080501/CourseModel.xsd
http://www.omg.org/spec/BPDM/20080501/importfile_commoninfrastructure.xsd
http://www.omg.org/spec/BPDM/20080501/InteractionProtocol.xsd
http://www.omg.org/spec/BPDM/20080501/InteractiveBehaviorModel.xsd
http://www.omg.org/spec/BPDM/20080501/xmi_infra.xsd
http://www.omg.org/spec/BPDM/20080501/VotingSample
http://www.omg.org/spec/BPDM/20080501/BPMNSamples_schema.xsd

http://www.omg.org/spec/BPDM/20080502
http://www.omg.org/spec/BPDM/20080502/xmi.xsd

Source document: BPDM Process Definitions Document without change bars (dtc/2008-05-10)
* Original file: XML schema and library (dtc/2008-05-14)

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.
Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. 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, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , MOF™ , OMG Interface Definition Language (IDL)™ , and OMG SysML™ 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

1 Scope....1
1.1 Business Process Modeling Notation (BPMN)....1
1.1.1 Target Audience and Use of BPDM....2
1.2 Other Common Business Benefits of BPDM....2
1.2.1 Carefully defined semantics....2
1.2.2 Saying just enough, but not too much....2
1.2.3 Improved Integration and Collaboration....2
1.2.4 Improved Agility....2
1.2.5 Business Processes supported by Service Oriented Architectures (SOA)....3
1.2.6 Better Return on I.T. Investment....3
1.3 Process Concepts Supported by BPDM....4
2 Conformance....4
2.1 BPDM Full Compliance....4
2.2 BPDM Collaboration Protocol Compliance....5
2.3 BPDM Orchestration Process Compliance....5
2.4 BPDM - BPMN Compliance....5
3 Normative References....5
4 Terms and Definitions....5
5 Additional Information....8
5.1 Acknowledgements....8
6 Metamodel and Notation Specification....8
6.1 Overview....8
6.2 Behavior Model....11
6.2.1 Introduction....11
6.2.2 Metamodel Specification....13
6.2.2.1 Behavior Model Diagram....14
6.2.2.2 Behavior Library: Events....15
6.2.2.3 Behavior Library: Behavior Occurrence....16
6.2.2.4 Behavior Library: 'Racing' Behavior....17
6.2.2.5 Behavior Library: 'Group Abort Behavior'....18
6.2.2.6 Behavior Event Condition Diagram....19
6.2.2.7 Behavior Step Group Diagram....19
6.2.2.8 Connected Part Binding Diagram....20
6.2.2.9 Behavior....20
6.2.2.10 Behavior Event Condition....21
6.2.2.11 Behavior Step....21
6.2.2.12 Behavior Step Group....22
6.2.2.13 Bindable Connection....22
6.2.2.14 Compound Behavioral Connection....22
6.2.2.15 Connected Part Binding....23
6.2.2.16 Event Monitor....23
6.2.2.17 Group Abort Connection....25
6.2.2.18 ImmediateSuccession....25
6.2.2.19 Race Connection....25
6.2.2.20 Succession....25
6.2.2.21 Instance: Abnormal End Event....26
6.2.2.22 Instance: Abnormal End....26

6.2.2.23 Instance: Abort Event....27
6.2.2.24 Instance: Abort....27
6.2.2.25 Instance: Behavior Library....28
6.2.2.26 Instance: Behavior Library....28
6.2.2.27 Instance: Behavior Occurrence....28
6.2.2.28 Instance: Enclosed Step....30
6.2.2.29 Instance: end/abort....30
6.2.2.30 Instance: Error Event....31
6.2.2.31 Instance: Error....31
6.2.2.32 Instance: Failure Event....32
6.2.2.33 Instance: Failure....33
6.2.2.34 Instance: Group Abort Behavior....33
6.2.2.35 Instance: group-step....34
6.2.2.36 Instance: ImportInfra....34
6.2.2.37 Instance: Normal End Event....35
6.2.2.38 Instance: Normal End....35
6.2.2.39 Instance: Racing Behavior....36
6.2.2.40 Instance: Racing Contestant....36
6.2.2.41 Instance: start/start....36
6.2.2.42 Instance: Step Group....37
6.2.2.43 Instance: Success Event....37
6.2.2.44 Instance: Success....37
6.3 Interactive Behavior Model....38
6.3.1 Introduction....38
6.3.2 Metamodel Specification....39
6.3.2.1 Interactive Behavior Diagram....39
6.3.2.2 Simple Interaction Binding Diagram....41
6.3.2.3 Message Diagram....41
6.3.2.4 End Message....41
6.3.2.5 Interaction....42
6.3.2.6 Interaction Role....42
6.3.2.7 Interactive Behavior....43
6.3.2.8 Interactive Part....43
6.3.2.9 Message....43
6.3.2.10 Message Flow....44
6.3.2.11 Received Intermediate Message....44
6.3.2.12 Sent Intermediate Message....45
6.3.2.13 Simple Interaction....46
6.3.2.14 Start Message....47
6.4 Activity Model....47
6.4.1 Introduction....47
6.4.2 Metamodel Specification....49
6.4.2.1 Activity Model Diagram....50
6.4.2.2 Activity Model Library: Simple Process instances....51
6.4.2.3 Activity Categories Diagram....51
6.4.2.4 Activity Model Library: Loop Happening instance....52
6.4.2.5 Embedded Process Diagram....53
6.4.2.6 Process Derivation Diagram....54
6.4.2.7 Role Realization Diagram....54
6.4.2.8 Abort Activity....54
6.4.2.9 Activity....55
6.4.2.10 Activity Loop....55
6.4.2.11 Actor....56
6.4.2.12 Conditional Loop....56
6.4.2.13 Embedded Process....57
6.4.2.14 Error Activity....58
6.4.2.15 Holder....58
6.4.2.16 LoopTestTime....59
6.4.2.17 Multi Instance Loop....59
6.4.2.18 MultiInstanceLoopOrdering....59
6.4.2.19 Performer Role....60

6.4.2.20 Process....61
6.4.2.21 Process Interaction Boundary....62
6.4.2.22 Processor Role....63
6.4.2.23 Role Realization....64
6.4.2.24 Simple Activity....64
6.4.2.25 Sub-Process Activity....64
6.4.2.26 Substitutable Derivation....65
6.4.2.27 Instance: Abort Process....66
6.4.2.28 Instance: Activity Library....66
6.4.2.29 Instance: Activity Loop Behavior....66
6.4.2.30 Instance: Error Process....66
6.4.2.31 Instance: Generalization....67
6.4.2.32 Instance: interationend-end....67
6.4.2.33 Instance: IterationEnd Event....67
6.4.2.34 Instance: IterationEnd....67
6.4.2.35 Instance: start-iterationend....68
6.5 BPMN Extensions....68
6.5.1 Introduction....68
6.5.2 Metamodel Specification....68
6.5.2.1 Adhoc Extension Diagram....69
6.5.2.2 Activity Extensions Diagram....69
6.5.2.3 Gateway Extension Diagram....70
6.5.2.4 BPMN Extensions Library: Compensate Process Instance....70
6.5.2.5 BPMN Extensions Library: BPMN Process Occurrence Instance....71
6.5.2.6 Sequence Flow Extension Diagram....71
6.5.2.7 Artifact Flow Extensions Diagram....72
6.5.2.8 Transaction Extensions Diagram....72
6.5.2.9 Compensation Extensions Diagram....72
6.5.2.10 Adhoc Process Directive....72
6.5.2.11 AdhocOrdering....73
6.5.2.12 Artifact Flow....73
6.5.2.13 Artifact Sequence Flow....73
6.5.2.14 Cancel Activity....74
6.5.2.15 Compensate Activity....75
6.5.2.16 Compensating Activity....75
6.5.2.17 Complex Decision....76
6.5.2.18 Complex Merge....76
6.5.2.19 Event Decision....77
6.5.2.20 Exclusive Decision....77
6.5.2.21 Exclusive Merge....78
6.5.2.22 Inclusive Decision....79
6.5.2.23 Inclusive Merge....80
6.5.2.24 Process Directive....80
6.5.2.25 Script Activity....80
6.5.2.26 Sequence Flow....80
6.5.2.27 Task....81
6.5.2.28 Terminate....81
6.5.2.29 Transaction....82
6.5.2.30 Instance: Cancel Event....82
6.5.2.31 Instance: Cancel Process....83
6.5.2.32 Links Instance: cancel-end....83
6.5.2.33 Instance: Cancel....83
6.5.2.34 Instance: Compensate Event....84
6.5.2.35 Instance: Compensate Process....84
6.5.2.36 Instance: compensate-end....84
6.5.2.37 Instance: Compensate....84
6.5.2.38 Instance: Compensation Library....85
6.5.2.39 Instance: Generalization....85
6.5.2.40 Instance: Process Occurrence....85
6.5.2.41 Instance: start-cancel....85
6.5.2.42 Instance: start-compensate....86
6.5.2.43 Instance: StartFromSequence....86

6.5.2.44 Instance: startseq-end....86
6.6 Interaction Protocol Model....86
6.6.1 Introduction....86
6.6.2 Metamodel Specification....87
6.6.2.1 Interaction Protocol....87
6.6.2.2 Compound Interaction....88
6.6.2.3 Compound Interaction Binding....89
6.6.2.4 Interaction Protocol....89
6.7 Class Hierarchies....89
6.7.1 Condition Hierarchy....90
6.7.2 Happening OverTime Hierarchy....90
6.7.5 Simple Interaction Hierarchy....92
6.7.6 Interactive Part Hierarchy....92
7 BPMN Notation Summary....93
7.1 Interaction Role Notation....93
7.2 Processor Role Notation....93
7.3 Horizontal Lane Notation....93
7.4 Vertical Lane Notation....94
7.5 Time Event Notation....95
7.6 Fact Change Notation....96
7.7 Course Event 'Error' Instance Notation....96
7.8 Course Event 'Cancel' Instance Notation....96
7.9 Course Event 'Iteration End'....97
7.10 Course Event 'Abort' Notation....97
7.11 Course Event 'Compensate' Instance Notation....97
7.12 Event Part : Start Notation....98
7.13 Event Part : Start with 'Time Event Condition' Notation....98
7.14 Event Part : Start with 'Fact Change Condition' Notation....98
7.15 Event Part : End Notation....99
7.16 Event Part : Error Notation....99
7.17 Event Part : Cancel Notation....99
7.18 Event Part : Abort Notation....100
7.19 Error Handling Notation....100
7.20 Activity Notation....101
7.21 Collapsed Sub-Process Activity Notation....101
7.22 Uncollapsed Sub-Process Activity Notation....102
7.23 Activity Loop Notation....102
7.24 Cancel Activity Notation or 'Cancel' Event Part....102
7.25 Error Activity Notation or 'Error' Event Part....103
7.26 Abort Activity Notation or 'Abort' Event Part....103
7.27 Compensate Activity Notation....104
7.28 Compensating Activity Notation....104
7.29 Event Monitor Notation....104
7.30 Event Monitor monitoring a Time Event Condition....105
7.31 Event Monitor monitoring a Fact Change Condition....105
7.32 Event Monitor monitoring a 'Compensate' Behavior Event Condition....106
7.33 Event Monitor monitoring a Compound Event Condition....106
7.34 Succession Notation....106

7.35 Event Decision Notation....107
7.36 Message Notation....107
7.37 Start Message Notation....107
7.38 End Message Notation....108
7.39 Sent Intermediate Message Notation....109
7.40 Received Intermediate Message Notation....110
7.41 Message Flow Notation....110
7.42 Artifact Sequence Flow Notation....111
7.43 Part Group Notation....111
7.44 Transaction Notation....112
7.45 Gateway Notation....112
7.46 Exclusive Split Notation....112
7.47 Exclusive Merge Notation....113
7.48 Parallel Split Notation....114
7.49 Parallel Join Notation....114
7.50 Inclusive Split Notation....115
7.51 Inclusive Merge Notation....116
7.52 Complex Decision Notation....116
7.53 Complex Join Notation....117
8 Non-normative Notation Summary....118
8.1 Process Diagram....118
8.2 Non-immediate Succession....118
8.3 Course Event 'Normal End' instance notation....118
8.4 Course Event 'Abnormal End' instance notation....119
8.5 Course Event 'Failure' Instance notation....119
8.6 Course Event 'Success' Instance Notation....119
8.7 Event Part : Normal End Notation....120
8.8 Event Part : Abnormal End notation....120
8.9 Event Part : Success Notation....120
8.10 Event Part : Failure Notation....121
8.11 Succession with Fact Change Condition....121
8.12 Succession with Time Event Condition....122
8.13 Interaction Flow between Activities and Statement Condition....122
8.14 Interaction Flow between Activities and Time Event Condition....123
8.15 Holder Notation....123
8.16 Compound Interaction Notation....123
8.17 Course Occurrence Diagram....124
8.18 Behavior Occurrence....125
8.19 Process Occurrence....126
9 BPDM–BPEL Mapping....127
9.1 General....127
9.1.1 Topological Considerations....127
9.1.2 Generator Model....127
9.1.3 Notational Conventions....127
9.2 Process....128
9.3 Start Event Mappings....128

9.4 End Event Mappings....129
9.5 Intermediate Events....130
9.6 Activities....134
9.7 Flows....138
9.8 Additional Constructs....140
9.9 References....142
10 Proof of Concept Language Mappings....143

List of Figures

Figure 1 - Dependencies of BPDM Packages....9
Figure 2 - Behavior Model Diagram....14
Figure 3 - Behavior Library: Events....15
Figure 4 - Behavior Library : Behavior Occurrence....16
Figure 5 - Behavior Library: 'Racing' Behavior....17
Figure 6 - Behavior Library: 'Group Abort Behavior'....18
Figure 7 - Behavior Event Condition Diagram....19
Figure 8 - Behavior Step Group Diagram....19
Figure 9 - Connected Part Binding Diagram....20
Figure 10 - Event Monitor monitoring a 'Compensate' Course Event Condition....24
Figure 11 - Event Monitor monitoring a Compound Event Condition....24
Figure 12 - Event Monitor monitoring a Fact Change Condition....24
Figure 13 - Event Monitor monitoring a Time Event Condition....24
Figure 14 - Event Monitor Notation....25
Figure 15 - Course Event 'Abnormal End' instance notation....26
Figure 16 - Event Part : Abnormal End notation....26
Figure 17 - Course Event 'Abort' Notation....27
Figure 18 - Event Part : Abort Notation....27
Figure 19 - Behavior Occurrence....30
Figure 20 - Course Event 'Error' Instance Notation....31
Figure 21 - Error Activity Notation or 'Error' Course Event Step....32
Figure 22 - Error Handling Notation....32
Figure 23 - Event Part : Error Notation....32
Figure 24 - Course Event 'Failure' Instance notation....33
Figure 25 - Event Part : Failure Notation....33
Figure 26 - Course Event 'Normal End' instance notation....35
Figure 27 - Event Part : Normal End notation....35
Figure 28 - Course Event 'Success' Instance notation....37
Figure 29 - Event Part : Success Notation....38
Figure 30 - Interactive Behavior Diagram....40
Figure 31 - Simple Interaction Binding Diagram....41
Figure 32 - Message Diagram....41
Figure 33 - End Message Notation....42
Figure 34 - Interaction Role Notation....42
Figure 35 - Message Notation....44
Figure 36 - Message Flow Notation....44
Figure 37 - Received Intermediate Message Notation....45
Figure 38 - Sent Intermediate Message Notation....46
Figure 39 - Start Message Notation....47
Figure 40 - Activity Model Diagram....50
Figure 41 - Activity Model Library: Simple Process instances....51
Figure 42 - Activity Categories Diagram....51
Figure 43 - Activity Model Library: Loop Happening instance....52
Figure 44 - Embedded Process Diagram....53
Figure 45 - Process Derivation Diagram....54
Figure 46 - Role Realization Diagram....54
Figure 47 - Abort Activity Notation or 'Abort' Behavioral Change Part....55
Figure 48 - Activity Notation....55
Figure 49 - Activity Loop Notation....56
Figure 50 - Collapsed Sub-Process Activity Notation....57
Figure 51 - Uncollapsed Sub-Process Activity Notation....58
Figure 52 - Error Activity Notation or 'Error' Behavioral Event Step....58
Figure 53 - Holder Notation....59

Figure 54 - Horizontal Lane Notation....60
Figure 55 - Vertical Lane Notation....61
Figure 56 - Process Diagram....62
Figure 57 - Interaction Role Notation....63
Figure 58 - Processor Role Notation....63
Figure 59 - Activity Notation....64
Figure 60 - Collapsed Sub-Process Activity Notation....65
Figure 61 - Uncollapsed Sub-Process Activity Notation....65
Figure 62 - Behavioral Event 'Iteration End'....67
Figure 63 - Adhoc Extension Diagram....69
Figure 64 - Activity Extensions Diagram....69
Figure 65 - Gateway Extension Diagram....70
Figure 66 - BPMN Extensions Library: Compensate Process Instance....70
Figure 67 - BPMN Extensions Library: BPMN Process Occurrence Instance....71
Figure 68 - Sequence Flow Extension Diagram....71
Figure 69 - Artifact Flow Extensions Diagram....72
Figure 70 - Transaction Extensions Diagram....72
Figure 71 - Compensation Extensions Diagram....72
Figure 72 - Artifact Sequence Flow Notation....74
Figure 73 - Interaction Flow between Activities and Statement Condition....74
Figure 74 - Interaction Flow between Activities and Time Event Condition....74
Figure 75 - Cancel Activity Notation or 'Cancel' Behavioral Event Step....75
Figure 76 - Compensate Activity Notation....75
Figure 77 - Compensating Activity Notation....76
Figure 78 - Complex Decision Notation....76
Figure 79 - Complex Join Notation....77
Figure 80 - Event Decision Notation....77
Figure 81 - Exclusive Split Notation....78
Figure 82 - Exclusive Merge Notation....79
Figure 83 - Inclusive Split Notation....79
Figure 84 - Inclusive Merge Notation....80
Figure 85 - Succession Notation....81
Figure 86 - Activity Notation....81
Figure 87 - Abort Activity Notation or 'Abort' Behavioral Change Part....82
Figure 88 - Transaction Notation....82
Figure 89 - Behavioral Event 'Cancel' Instance Notation....83
Figure 90 - Event Part : Cancel Notation....83
Figure 91 - Behavioral Event 'Compensate' Instance Notation....84
Figure 92 - Interaction Protocol....88
Figure 93 - Compound Interaction Notation....89
Figure 94 - Condition Hierarchy....90
Figure 95 - Happening OverTime Hierarchy....90
Figure 96 - Event Hierarchy....91
Figure 97 - Behavioral Step Hierarchy....91
Figure 98 - Simple Interaction Hierarchy....92
Figure 99 - Interactive Part Hierarchy....92
Figure 100 - Interaction Role Notation....93
Figure 101 - Processor Role Notation....93
Figure 102 - Horizontal Lane Notation....94
Figure 103 - Vertical Lane Notation....95
Figure 104 - Time Event Notation....95
Figure 105 - Fact Change Notation....96
Figure 106 - Course Event 'Error' Instance Notation....96
Figure 107 - Course Event 'Cancel' Instance Notation....96
Figure 108 - Course Event 'Iteration End'....97
Figure 109 - Course Event 'Abort' Notation....97
Figure 110 - Course Event 'Compensate' Instance Notation....97
Figure 111 - Event Part : Start Notation....98
Figure 112 - Event Part : Start with 'Time Event Condition' Notation....98
Figure 113 - Event Part : Start with 'Fact Change Condition' Notation....98
Figure 114 - Event Part : End Notation....99
Figure 115 - Event Part : Error Notation....99

Figure 116 - Event Part : Cancel Notation....99
Figure 117 - Event Part : Abort Notation....100
Figure 118 - Error Handling Notation....100
Figure 119 - Activity Notation....101
Figure 120 - Collapsed Sub-Process Activity Notation....101
Figure 121 - Uncollapsed Sub-Process Activity Notation....102
Figure 122 - Activity Loop Notation....102
Figure 123 - Cancel Activity Notation or 'Cancel' Event Part....103
Figure 124 - Error Activity Notation or 'Error' Event Part....103
Figure 125 - Abort Activity Notation or 'Abort' Event Part....103
Figure 126 - Compensate Activity Notation....104
Figure 127 - Compensating Activity Notation....104
Figure 128 - Event Monitor Notation....105
Figure 129 - Event Monitor monitoring a Time Event Condition....105
Figure 130 - Event Monitor monitoring a Fact Change Condition....105
Figure 131 - Event Monitor monitoring a 'Compensate' Behavior Event Condition....106
Figure 132 - Event Monitor monitoring a Compound Event Condition....106
Figure 133 - Succession Notation....106
Figure 134 - Event Decision Notation....107
Figure 135 - Message Notation....107
Figure 136 - Start Message Notation....108
Figure 137 - End Message Notation....108
Figure 138 - Sent Intermediate Message Notation....109
Figure 139 - Received Intermediate Message Notation....110
Figure 140 - Message Flow Notation....110
Figure 141 - Artifact Sequence Flow Notation....111
Figure 142 - Part Group Notation....111
Figure 143 - Transaction Notation....112
Figure 144 - Gateway Notation....112
Figure 145 - Exclusive Split Notation....113
Figure 146 - Exclusive Merge Notation....114
Figure 147 - Parallel Split Notation....114
Figure 148 - Parallel Join Notation....115
Figure 149 - Inclusive Split Notation....115
Figure 150 - Inclusive Merge Notation....116
Figure 151 - Complex Decision Notation....116
Figure 152 - Complex Join Notation....117
Figure 153 - Process Diagram....118
Figure 154 - Non Immediate Succession....118
Figure 155 - Course Event 'Normal End' instance notation....119
Figure 156 - Course Event 'Abnormal End' instance notation....119
Figure 157 - Course Event 'Failure' Instance notation....119
Figure 158 - Course Event 'Success' Instance notation....119
Figure 159 - Event Part : Normal End notation....120
Figure 160 - Event Part : Abnormal End notation....120
Figure 161 - Event Part : Success Notation....121
Figure 162 - Event Part : Failure Notation....121
Figure 163 - Succession with Fact Change Condition....121
Figure 164 - Succession with Time Event Condition....122
Figure 165 - Interaction Flow between Activities and Statement Condition....122
Figure 166 - Interaction Flow between Activities and Time Event Condition....123
Figure 167 - Holder Notation....123
Figure 168 - Compound Interaction Notation....123
Figure 169 - Course Occurrence Diagram....124
Figure 170 - Behavior Occurrence....125
Figure 171 - Process Occurrence....126

1 Scope

The "Business Process Definition Metamodel" (BPDM) is a framework for understanding and specifying the processes of an organization or community. Business processes have been at the heart of business and technology improvement under the guise of many terms and methodologies, such as: Business Process Engineering or Re-Engineering, Business Process Management, Business Process Execution, Total Quality Management, Process Improvement, Business Process Modeling, and Workflow. Similar and related concepts such as Service Oriented Architectures, Enterprise Application Integration, Flowcharts, Data Flows, Activity Diagrams, Role/Collaboration Modeling, and Modeling and Simulation serve to enable and describe business processes.

This heritage of process related approaches has provided substantial benefit to public and private institutions and is one of the factors that has allowed the modern enterprise to grow and prosper. This same heritage has also caused some confusion in how these various approaches and solutions do or do not work together and how to leverage them for a coherent and integrated solution. As of now there is a substantial asset of business process descriptions, notations, implementations, and machinery but many of these are islands – islands of a particular technology, methodology, or notation.

BPDM provides the capability to represent and model business processes independent of notation or methodology, thus bringing these different approaches together into a cohesive capability. This is done using a "meta model"[1] – a model of how to describe business processes – a kind of shared vocabulary of process with well defined connections between terms and concepts. This meta model captures the meaning behind the notations and technologies in a way that can help integrate them and leverage existing assets and new designs. The meta model behind BPDM uses the OMG "Meta Object Facility" (MOF)[2] standard to capture business processes in this very general way, and to provide an XML syntax for storing and transferring business process models between tools and infrastructures. Various tools, methods, and technologies can then map their way to view, understand, and implement processes to and through BPDM.

To achieve this goal, BPDM supports two fundamental and complementary views of process – "Orchestration" and "Choreography":

- Orchestration concepts in BPDM are represented through sequences of *"Activities"* that produce results with branching and synchronization. Orchestration is typically represented as flow charts, activity diagrams, swim lanes, or similar notations of one task or activity following another. The orchestration of processes describes what happens and when in order to better manage a process under the authority of some entity.

- Choreography describes how semi-independent and collaborating entities work together in a process, each of which may have their own internal processes. Choreography captures the *interactions* of *roles* with well defined responsibilities within a given process. Choreography is the basis for the *Service Oriented Architecture (SOA)* paradigm and helps to keep the enterprise loosely coupled and agile. The choreography of a process focuses on the responsibilities and interactions that ultimately provide value without necessarily requiring any coordinating authority.

In business process modeling, choreography and orchestration are effectively two sides of the same coin. BPDM joins orchestration and choreography into a unified and coherent model.

1.1 Business Process Modeling Notation (BPMN)

BPMN has gained recognition as a flexible and business-friendly notation for process orchestration. ***BPDM provides an explicit metamodel and serialization mechanism for BPMN concept*s**. By integrating BPMN and BPDM both the underlying model and notation for process orchestration is covered by an integrated set of standards. The notation for choreography, BPMN diagram interchange and the normative relationship to runtime technologies such as BPEL is planned to be part of subsequent standards.

---

1 Meta models - http://en.wikipedia.org/wiki/Meta_model

2 OMG Meta Object Facility - http://www.omg.org/mof/

1.1.1 Target Audience and Use of BPDM

At its core, BPDM provides interoperability across tools, so that different tools can depict or utilize a process definition in different ways yet work together for the ultimate benefit of the enterprise. For example, If Vendor A and Vendor B both support BPDM as their process exchange mechanism, then, a BPMN drawing created using Vendor A's modeling tool could then be opened and executed using Vendor B's business process management system. Therefore, BPDM is a technology specification for vendors to use to define how they serialize or exchange their process depictions, allowing for industry interoperability. For most business analysts and process users, this is all they really need to know about BPDM. What BPDM support means is that your process assets are not locked into a particular tool or notation; they are assets that can work across a wide range of tools and solutions.

1.2 Other Common Business Benefits of BPDM

1.2.1 Carefully defined semantics

When diagrams are used to aid human to human communications a certain amount of "fuzziness" in what those notations mean can be acceptable, since explanations often clear up any misunderstandings. When *processes are specifications* for what people, organizations, or I.T. systems should do, those specifications must be clear and precise. Particular attention has been paid in BPDM to make sure that the semantics behind the notations and models are well defined, consistent and sufficient to represent most normal forms of business processes. BPDM is sufficiently precise to model behavioral events (starting, ending, aborting, etc) of processes that allows them to be ordered in time, and have their effects on each other precisely modeled. Formal methods[3], based on logic, are utilized to verify this precision. The precise semantics of BPDM makes sure that processes will be accurately communicated to man and machine.

1.2.2 Saying just enough, but not too much

Specifying a business process can be a double-edged sword. Say too little and the process may be unpredictable, inconsistent, wasteful, and not fit into the rest of the business (or the business of partners). Say too much and the process can be a strangle-hold, preventing creativity, agility, and optimization. BPDM can't enforce this artful balance, but it can enable it; the basis of which is separation of concerns – separating the intended outcome of a process from how that outcome is achieved. Where appropriate; substantial detail can be specified for how to achieve a goal, in other cases only the "contract" is specified – the contract says what is to be accomplished without saying how. Many of the established methods do not provide well for this separation of concerns and therefore over specify or under specify a process. BPDM provides for separation of concerns, well defined contracts, and multiple options for implementing a process that corresponds to its contracts.

1.2.3 Improved Integration and Collaboration

The successful modern enterprise is defined by two basic capabilities; the ability to be agile and the ability to collaborate. Both capabilities are served by "loosely coupling" the business and the technologies that serve it. This means that tightly coupled and monolithic processes are barriers to success. A business process design better serves the enterprise by making it easy to collaborate with other organizations, regardless of their processes. It should be easy to outsource, insource or change the way a part of the organization works without undue impact on the rest of the organization or business partners. The integration of orchestration and collaboration as well as the separation of process contract from its realization serve this goal of loose coupling.

1.2.4 Improved Agility

Agility is required to respond to external drivers, internal needs and the constant impact of legislation and technology change. In today's' world – agility is survival. The combination of well defined business processes that provide for separation of concerns with Model Driven Architecture (MDA)® [4] provide the exciting possibility of being able to design, redesign and deploy new processes quickly and with minimal overhead – the enterprise is not locked in to legacy technologies and processes. BPDM provides the business focused model that can be part of the specification of the process for people, in terms of process "play books" and instructions, and for technologies, in

[3] Process Specification Language (PSL) - http://www.mel.nist.gov/psl/

[4] Model Driven Architecture (MDA)®, is a trademark of the Object Management Group – http://www.omg.org/mda

terms of web services, workflows, and process execution engines. In addition BPDM is technology independent – any number of technical approaches may be used to help realize or support a business process. The BPDM model is a model of the business, not the technology – MDA helps join these two viewpoints.

1.2.5 Business Processes supported by Service Oriented Architectures (SOA)

SOA has become recognized as the leading architectural approach to business and technical agility and integration. SOA structures the enterprise and supporting technologies based on *services* that are provided or consumed by collaborating entities. This service oriented approach applies to both the business – in terms of how one business or business unit serves another, and to the technologies – in terms of how application components work together by providing and using software services. BPDM describes the business side of SOA in terms of choreography (above) that can then be mapped to the software components that assist those business processes. This process centric SOA approach provides for agility, loose coupling, and a better tie between business and technology. SOA helps support both the agility and collaboration goals of BPDM.

1.2.6 Better Return on I.T. Investment

The net result of separation of concerns, support for collaboration, and enhanced agility is that I.T. investments have better return. This return is realized by directly supporting business needs as identified in the business processes and by supporting reuse of services, components, and supporting infrastructure across the enterprise and across marketplaces. Since investments are more reusable, their return is not limited to a single project. Since investments are directly tied to business needs, their business benefit can be measured. Since investments support agility and collaboration, they can have bottom-line impact.

1.3 Process Concepts Supported by BPDM

BPDM integrates multiple process approaches and notations, which are summarized as follows. BPDM provides integrated and consistent support for the semantics of:

- All BPMN notation concepts
- Processes, activities, tasks, and sub-processes
- Workflow
- Sophisticated control of alternatives and parallel processes
- Conditional execution paths
- Signals and events
- Time-based events and conditions
- Events based on change in data or external conditions
- Integration with rules and rules engines through event-based semantics
- Process groups and swim-lanes
- Transactions, rollback, and compensation
- Process data and data flow
- Artifacts and artifact production and dependencies
- A combination of human and automated process participants
- Service Oriented Architectures and business services
- Resource and entity selection
- Roles, responsibilities, and collaborations
- Bi-directional and composite interactions between entities
- Automated execution with MDA and process execution engines such as BPEL (See non-normative mapping to BPEL)
- Interaction protocols, services, and design by contract
- Composite processes
- UML activity, collaboration, and interaction diagram concepts
- Process specialization, derivation, and refinement.

In summary, BPDM standardizes the underlying semantics, model, and exchange mechanisms to improve the efficiency, agility, and collaboration of public and private enterprises through the precise and integrated definition of business processes.

2 Conformance

The following levels of compliance are defined for BPDM in relation software. For the following compliance points the interpretation of the phrase “to process a model” will depend on the functionality of the software as follows:

- If the software reads process models, to “process the model” will include reading a BPDM model compliant with the MOF-2 XMI for BPDM included as part of this specification.

- If the software writes process models, to “process the model” will include writing a BPDM model compliant with the MOF-2 XMI for BPDM included as part of this specification.

- If the software executes or otherwise interprets process models, to “process the model” will include executing or interpreting the model in accordance with the semantics as defined in this document.

2.1 BPDM Full Compliance

An implementation is fully compliant if it can process a model that utilizes all BPDM metamodel concrete concepts, not necessarily including those defined in the “BPMN Extensions” package.

2.2 BPDM Collaboration Protocol Compliance

An implementation is BPDM protocol compliant if it can process a collaboration protocol model that utilizes all concrete concepts for representation of a collaboration protocol as specified in the "Interaction Protocol Process Model" package and all included packages.

2.3 BPDM Orchestration Process Compliance

An implementation is BPDM protocol compliant if it can process an orchestration model that utilizes all concrete concepts for representation of an orchestration process as specified in the "Activity Model" package and all included packages.

2.4 BPDM - BPMN Compliance

An implementation is BPMN compliant if it can process a model that utilizes all concrete concepts for representation of a process as specified in Section 6.5, 6.6, 6.7, 6.8 and 6.9. Each of these sections provides detailed mappings of the BPMN notation constructs on to the BPDM metamodel. Section 7 provides a mapping summary of BPMN notation constructs to BPDM metamodel elements.

3 Normative References

[BPEL11] ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf
[BPEL20] http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.pdf
[BPMN] http://www.omg.org/cgi-bin/apps/doc?dtc/07-06-03.pdf
[BPM-06-02] http://is.tm.tue.nl/staff/wvdaalst/BPMcenter/reports/2006/BPM-06-02.pdf
[RFC2119] http://www.ietf.org/rfc/rfc2119.txt

4 Terms and Definitions

Activity

An **Activity** is a kind of **Behavior Step** that activates a **Behavior** (it operates over time) in the context of a **Process**. It can:

- be ordered in time by **Succession**
- operate under the responsibility of a **Performer Role**
- activate a sub-processe or be a simple task that start and stop

An **Activity** is also an **Interactive Part** that receives its inputs and outputs through **Interactions** coming from other **Interactive Part**s in the **Process** (**Activity**, **Interaction Role**, **Performer Role**, **Holder**).

Actor

An **Actor** is an entity that is responsible for the execution of duties specified by a **Performer Role**
Further sub-type of **Actor** will be defined in specifications such as the Organizational Structure Metamodel (OSM) to add specific requirements such as and can as having certain skills or budget.

Performer Role

A **Performer Role** is a **Part Group** that takes responsibility of performing activities in the process. Being an **Interactive Part**, a **Performer Role** also has responsibilities to fulfill **Interactions** that it is involved with other **Performer Roles** or with **Interaction Roles** at the boundary of the **Process**. A **Performer Role** is a **Typed Part** for specifying **Actor** that can play the role at process enactment.

A **Performer Role** can be decomposed into sub **Performer Role** to delegate responsibility for a subset of its activities or interactions. A **Performer Role** may have a realization as defined by a **Role Realization** that further specifies how the **Performer Role** will meet its responsibilities.

Process

A **Process** is a kind of **Interactive Behavior** that describes specific **Activity**(ies) to be performed, **Interaction**s to be undertaken during its execution under the authority of a **Processor Role** (or **delegated performer role**s). The process owns the set of activities to be performed as well as the **Condition**s on when such activities will be performed and by which performer role. The process also owns the set of **Interactive Part**s that define the flow of information and other resources between activities, **Performer Role** and **Interaction Role**s.

A specific **Interaction Role** defines the set of **Interaction**s the process is responsible of: its is the **Process Interaction Boundary**. The set of **Interaction**s attached to the **Process Interaction Boundary** defines the inputs and outputs of the process.

A Process may utilize sub-processes with a **Sub-Process Activity** as well as be used in the context of other processes in the same way.

Behavior Step

Behavior Steps is a kind of **Happening Part** which typed is a **Behavior**. This enables it to "invoke" other **Behavior** and to build **Behavior** composites (made of sub- **Behaviors**).

Behavior Step Group

A **Behavior Step Group** is a kind of **Part Group** that is also a **Behavior Step** typed by the **Behavior Occurrence** in user models (M1). This gives a group of **Behavior Steps** as a whole the capacity to produce start and end changes playing the standard behavioral change parts, such as **Start** and **End**. For example, most process languages have a way of modeling sub-processes without defining a separate process. This is a **Behavior Step Group**.

Event Monitor

An **Event Monitor** is a kind of **Behavior Step** that monitors the occurrence of an **Event Condition** and that has an effect on the course of a **Behavior**. For instance, an **Event Monitor** can be used to react to the **Abort Event** of a specific **Course**.

Interaction

An **Interaction** is a **Behavior Step** that is also a **Part Connection** , enabling **Interaction** to have start and end changes, and be ordered in time.

An **Interaction** can be either a simple **Simple Interaction** or a set of combined **Simple Interaction**s: a **Compound Interaction**. Ultimately, an **Interaction** is realized by the exchange of **Simple Interaction**s between its **Interactive Part**s.

Interaction Role

An **Interaction Role** is an **Interactive Part** where the individuals playing the part are in the environment context where the **Behavior** is used. For example, the customer is an **Interaction Role** in a behavior for delivering a product.

Simple Interaction

A **Simple Interaction** is a kind of **Interaction** in which something is "transferred" from individuals playing one interactive part to individuals playing another interactive part. For example, a document, phone number, or package may be transferred from one department to another in a company. The transferred items must conform to a **Type** specified by the **Simple Interaction**. A **Simple Interaction** can have an **Expression** to change the item that arrives at the target based on the item flowing from the source. For example, a transformation may retrieve the zip code

from an address flowing from the source to deliver the zip code to the target.
Simple Interactions in user (M1) models are always typed by the **Behavior Occurrence** (see user library **Behavior Library**). This gives them the standard **Event Parts**, such as for start and end, so the **Simple Interactions** can be ordered within an **Interaction Protocol**. This is different from the type of thing transferred.

Simple Interactions can refer to **Simple Interactions** inside the **Interactive Parts** being connected. This means the transferred thing is passed along through chains of **Simple Interactions** from inside to outside the parts, or the other way.

Interaction Protocol

An **Interaction Protocol** is a kind of **Interactive Behavior** where **Behavior Steps** are **Interactions** that occur between **Interaction Roles**. The set of **Interaction**s defines the purpose of the **Interaction Protocol**.

Condition

A **Condition** is a **Boolean ValueSpecification** that constrains some element in the models. Conditions are true if their descriptions hold in the current state of the world, possibly including executions, and false otherwise.

Course

A **Course** is an ordered **Succession** of **Happening Part**s. A **Course** is a **Composite** that has connections representing that one part of the course "follows" another in time, and possibly establishes constraints on such followings (**Succession**).

Event

An **Event** is a **Happening** for dynamic entities occurring at a point in time.

Event Condition

An **Event Condition** is a **Condition** for specifying that an **Event** must occur in the context of a particular **Happening Over Time** for the condition to hold. For instance, a condition can be on the eruption (instance of **Event**) of a particular volcano (instance of **Happening Over Time**).

Event Part

An **Event Part** identifies **Event** (such as **Start Event** or **End Event**) for an individual **Course**. An **Event Part** is also a **Happening Part**, enabling it to be connected by **Successions**.

Gateway

A **Gateway** is a kind of **Course Part** representing potentially complex specifications of how dynamic individuals playing **Happening Part**s are ordered in time. The particular specifications are given in subtypes. At runtime, **Gateways** don't have any execution trace.

Succession

A **Succession** is a **Directed Part Connection** that organizes **Course Parts** in series in the context of a **Course**. A **Succession** indicates that one **Course Part** "follows" another in time, and possibly establishes constraints on such followings. It can order the **Event Part** of its **Happening Parts** such as their **Start** or **End**.
Succession allows any combination of **Event Part** to be connected.

End -> Start
Start -> Start
Start -> Abort
etc.

A **Succession** doesn't need to have **Happening Part** on its ends, it can have untyped course parts also, such as **Gateway**, but it must have something on each end. For convenience, a **Succession** that does not specify **source**

event part or **target event part** will have the same effect as a **Succession** where these are respectively the **End** and **Start**.

Time Event

A **Time Event** specifies a point in time that is a source of interest.

Time Event Condition

A **Time Event Condition** is a kind of **Event Condition** that is based on the occurrence of a **Time Event**. A **Time Event Condition** is specified by referring to a **Clock**.

5 Additional Information

5.1 Acknowledgements

The following companies submitted this specification:

- Adaptive
- Axway Software
- Borland Software
- Model Driven Solutions
- EDS
- Lombardi Software
- MEGA International
- Unisys

The following companies and organizations support this specification:

- BPM Focus
- U.S. National Institute of Standards and Technology (NIST)

6 Metamodel and Notation Specification

This section presents the normative specification for business process definition metamodel, including its BPMN based notation. It begins with an overview of the BPDM metamodel structure followed by a description of each sub-package.

6.1 Overview

The **Business Process Definition MetaModel** package contains the models for orchestration (including BPMN) and choreography, and their performance, enactment, and execution. It has six subpackages grouped into two categories:

- **Common Behavior Model** for the aspects of dynamics in common between orchestrations and choreography (**Behavior Model**, and **Interactive Behavior Model**).

- **Activity Model** (including BPMN Extensions) for orchestration and **Interaction Protocol Model** for choreography.

The **Business Process Definition MetaModel** package imports the **Common Infrastructure** package which provides the framework that ties the other models to performance, enactment, and execution (Abstractions, Composition Model, Course Model and Condition Model).

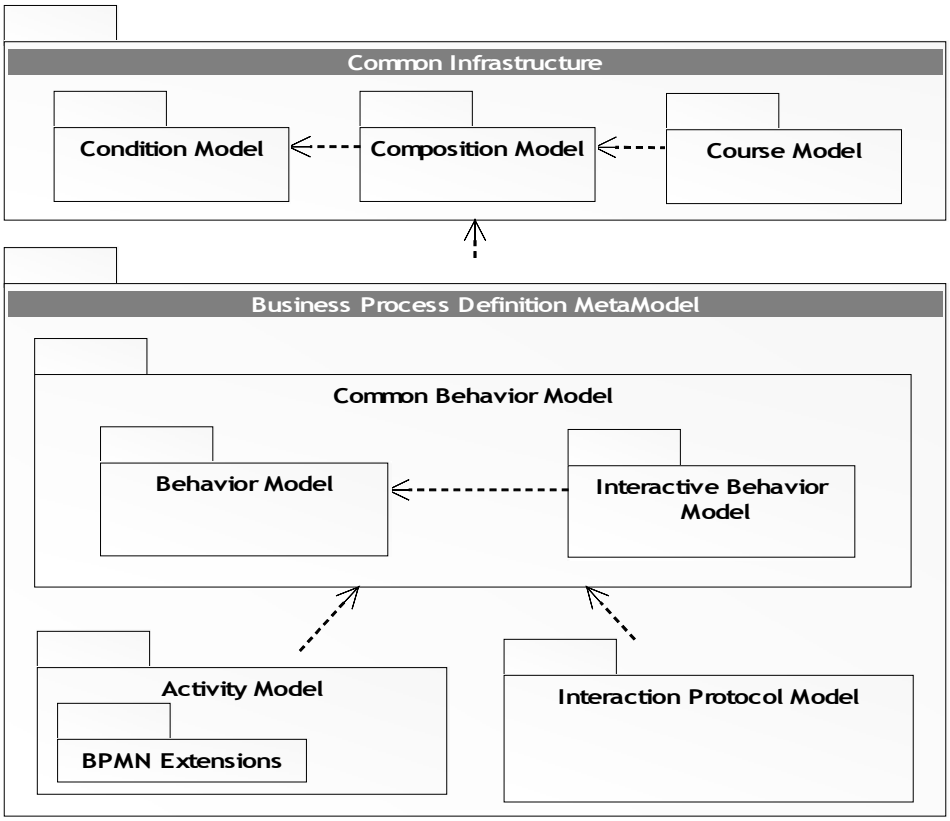


Figure 1 - Dependencies of BPDM Packages

Package	Comment
Business Process Definition MetaModel	The Business Process Definition MetaModel package contains the models for orchestration (including BPMN) and choreography, and their performance, enactment, and execution. It has six subpackages grouped into two categories: • Common Behavior Model for the aspects of dynamics in common between orchestrations and choreography (Behavior Model, and Interactive Behavior Model). • Activity Model (including BPMN Extensions) for orchestration and Interaction Protocol Model for choreography. The Business Process Definition MetaModel package imports the Common Infrastructure package which provides the framework that ties the other models to performance, enactment, and execution (Abstractions, Composition Model, Course Model, and Condition Model).
Common Infrastructure	The Common Abstractions package is the framework that ties the other models to performance, enactment, and execution (Composition Model, Course Model and Condition Model).
Composition Model	The Composition Model is a framework for relating metamodels to the real world entities they ultimately represent. It facilitates integration with business process runtimes and rule engines, as well as uniform performance, enactment, and execution across business process

	management suites. The Composition Model enables users and vendors to build libraries of orchestrations and choreographies, including specialization of some orchestrations or choreographies from others. It also enables users and vendors to define their own frameworks for recording data about ongoing orchestrations and choreographies, for example, how long they have been going, who is involved in them, and what resources they are using.
Course Model	The Course Model extends the Composition Model to connect parts in time (Succession). For example, a succession connects one step in a process to another to indicate that the second step happens after the first. The same applies to messages in choreography.
Common Behavior Model	The Common Behavior Model includes elements shared by all process oriented behavior models.
Behavior Model	The Behavior Model enables Behavior Steps to be ordered in time as parts of other Behavior Step (see the Course Model). Vendors and users can define their own execution patterns with connections between these Behavior Steps. The model predefines a specific connection for races, where Behavior Steps start at the same time and abort each other when the first finishes. It also defines a Behavior Event Condition for detecting lifecycle events in behavioral happenings. The Behavior Model is the most specialized model in the Business Process Definition MetaModel that still covers all of processes and interactions (orchestration and choreography, see the Activity and Interaction Protocol Models).
Interactive Behavior Model	The Interactive Behavior Model enables interactions to be treated like any other step in a Behavior, ordered in time, with start and end events. The model is the basis for flows between Behavior Steps and between participants in a choreography (see the Activity Model and the Interaction Protocol Model). The Interactive Behavior Model is the most specialized model in the Business Process Definition MetaModel that still has elements in common between processes and choreographies.
Activity Model	The Activity Model is for capturing orchestrations in way that facilitates modification as boundaries of process of business change, for example, due to insourcing, outsourcing, mergers, and acquisitions. It uses interactions to represent inputs and outputs, enabling choreographies to be specified between the process and its environment, as well as between the performers responsible for steps in the process. The Activity Model is the basis for the BPMN model in BPDM (see the BPMN Extensions).
BPMN Extensions	The BPMN Extension provides additions to the Activity Model for BPMN. These provide BPMN names for special usages of the Business Process Definition MetaModel concepts and additional functionality specific to BPMN.
Interaction Protocol Model	The Interaction Protocol Model is for capturing choreographies. It enables interactions to be grouped together into larger, reusable interactions. For example, an interaction that exchanges goods between companies might be used with other interactions within a larger protocol representing a partnership of the companies. This protocol might be adopted by a standards body and reused between many pairs of companies. The interactions in a protocol may be simple interactions that have no sub-interactions, or may be other protocols.
Condition Model	The Condition Model is for specifying boolean expressions that constrain model elements or capture statements. It defines specialized conditions that are represented as free text, as expressions with particular results, and as boolean combinations of other conditions.

6.2 Behavior Model

6.2.1 Introduction

The Behavior Model extends the Course Model with a number of common behavior modeling constructs, and provides for vendor and user defined extensions. Vendors and users can define their own execution patterns with connections between these happening parts. The model predefines a specific connection for races, where behaviors start at the same time and abort each other when the first finishes, and for part groups that abort the steps inside them. It extends the events and event parts of the Course Model, for example, when behaviors are aborted. It also defines an event condition for detecting lifecycle events in behaviors. The Behavior Model is the most specialized model in BPDM that still covers all of orchestration and choreography (see the Activity Model and Interaction Protocol Model).

The Behavior Model introduces:

- Courses with parts that behaviors play (Behavior and Behavior Steps). These enable reuse of behaviors. A behavior orders subbehaviors according to their course events, such as when they start and end (see the Course Model).
- A taxonomy of events specializing starting and ending events from the Course Model, for example, for aborting and erroring. These play event parts from a taxonomy subsetted from the start and end parts in the Course Model, where the event parts are of Behavior Occurrence, a specialization of Course Occurrence.
- Connections for behavior steps that establish execution rules for connected steps (Compound Behavioral Connection). One of these is a connection between steps that all start at the same time, and where the first one to finish aborts the others (Race Connection and Race Behavior). Another connects groups that can abort their enclosed steps (Group Abort Connection and Group Abort Behavior).
- Behavior steps for monitoring events, such as changes in time, facts, or behavior (Event Monitor). For example, an event monitor can detect the passing of a certain point in time, a change in the truth of a statement due to changes in facts, and the completion of a happening, such as the arrival of a message.
- Groups of behavior steps (Behavior Step Group), where the group has its own event parts, such as for starting and ending.

Behaviors are Courses with Behavior Steps, which are Happening Parts where the type is a Behavior. This enables behaviors to "invoke" and order other behaviors in time, as in the steps of a process model and or the interactions in choreography. For example, the steps in a selling process are behavior steps played by behaviors such as packing and shipping. Individual selling processes (M0 performances, enactments, or executions of selling) can have a behavior step played by an individual (M0) packing behavior and another behavior step played by an individual shipping behavior.

A user (M1) library in the Behavior Model adds:

- Behavior Occurrences, a specialization of Course Occurrence (see the Course Model), and generalization of all M1 behavior models, including all orchestration and choreography models. All individual (M0) behavior occurrences conform to Behavior Occurrence, which is the most abstract M1 model of behaviors.

- A taxonomy of events generalized by the start and end events in the Course Model. End at M1 generalizes normal and abnormal events. Normal events generalize success and failure events, indicating whether an M0 behavior fulfills its purpose or not. Abnormal events generalize abort and error events. Aborting means an M0 behavior is terminated by an external source. Erroring means an M0 behavior terminates itself due to conditions it is not prepared to handle. Abnormal ending may involve cleanup, but this must be completed before the end of the behavior.

- A taxonomy of event parts subsetting the start and end event parts in the Course Model. The library event parts are subsetted to align with the subclassing of event types above, which means events playing the subsetting parts also play the subsetted parts.[5] For example, an event playing an abort part on an M0 behavior also plays the abnormal end and end parts on the same individual course. Each individual (M0) behavior occurrence will have at most one individual event conforming to the event types in the library. For example, there is at most one abort event for each individual course occurrence.

[5] Part and property subsetting are analogous to generalization, see the Composition Model.

- Behaviors for compound behavioral connections, see below.

Event Monitors are Behavior Steps that detect events, including changes in time, facts (see the Course Model), or behavior. Event monitors in user models (M1) are always typed by Behavior Occurrence or are subtypes of it that have no behavior steps. Successions can order event monitoring steps. For example, a process can perform one step, then perform a time event monitoring step to wait for a certain duration to elapse, then perform another step. This is enabled by event monitors at M1 being typed by Behavior Occurrence, to define the standard event parts, for example start part and end part.

Connected Part Bindings are Elements specifying that individuals playing the part at an end of a connection also play a part within the connection. For example, one of the interactions between businesses in a choreography might be a sub choreography composed of many communications between the businesses. Businesses playing a particular role in the larger choreography also play one of the roles in the sub choreography. Bindable Connections are defined just to categorize those connections that can carry part bindings. The player is part of the composite owning the bindable connection. The played is part of the bindable connection. The binding requires the (M0) individuals playing these parts to be the same. They are found by navigating from an individual composite, to the player individuals, and to the played individuals in the connection part of the same composite. The two sets of individuals found this way must be exactly the same. Connected part bindings are different from connections because part bindings are about which individuals are playing certain parts in a whole, whereas connections are about links between the individuals themselves due to playing parts in the whole. As a convenience, it is assumed that a connection typed by a composite that has only one (non-connection) part implies bindings where that one part is played by all the parts at all the ends of the connector. This is useful for symmetrical connectors (see Race Connector below for an application).

Compound Behavioral Connections are Connections between behavior steps that are also Typed Parts, enabling connections to reuse the same composite for connecting steps. BPDM defines two kinds of compound behavioral connections:

- Race Connections are Compound Behavioral Connections that are always typed by Race Behavior, an M1 instance of Behavior defined in the Behavior user (M1) library. Race Behavior ensures that all the behavior steps connected by Race Connection start at the same time, and that the first one to finish aborts the others. Race Behavior contains:

    - One step, called the Contestant, which is bound to all the steps connected by the M1 race connection (see Connected Part Binding above). This ensures that all the contestants are treated the same way.

    - Two immediate successions connecting the Contestant to itself. One succession refers to the start part of the Contestant on both ends (see the Happening Model), specifying that all the contestant behaviors start at the same time. The other succession has the finish part on one end and the abort part on the other, specifying that any contestant happening that finishes will be accompanied by a simultaneous abort of the others. This succession has the Irreflexive condition applied (see the Composition Model), to prevent the finishing contestant from aborting itself.

      When a race connection is created between behavior steps, it implies part bindings between the connected steps and the Contestant in Race Behavior, with Contestant on the played end (see Connected Part Binding above). The part bindings ensure that any individual M0 happening playing the connected steps will also play the Contestant, establishing the start-start and finish-abort successions between the connected steps, and the temporal constraints on the individual happenings. The Race Behavior above can be the type for any connector that is also a typed part, but Race Connection is always typed by Race Behavior, for convenience.

- Group Abort Connections are Compound Behavioral Connections that are always typed by Group Abort Behavior, an M1 instance of Behavior defined in the Behavior user (M1) library. It is applied to behavior step groups and their enclosed steps to ensure that the steps are aborted when the group is. Group Abort Behavior contains:

    - Two steps, one for the group and one for its enclosed steps (Step Group and Enclosed Step). The first is bound to an M1 behavior step group and the second to each step in the group (see Connected Part

Binding above).

- One immediate succession between the two steps above. The source is Step Group and the target is Enclosed Step. It refers to the abort event part on both ends, specifying that any group behavior that aborts will be accompanied by a simultaneous abort of the enclosed behaviors.

  When a group abort connection is created between a behavior step group and its steps, it implies a part binding between Step Group in the Group Abort Behavior and the connected group, with Step Group on the played end (see Connected Part Binding above). Similarly, it implies bindings between Enclosed Step and the steps in the group. The part bindings ensure that any individual M0 happening playing the connected group will also play the Step Group, and any individual playing the connected steps will also play the Enclosed Step, establishing the abort-abort successions between the connected group and steps, and the temporal constraints on the individual happenings. The Group Abort Behavior above can be the type for any connector that is also a typed part, but Group Abort Connection is always typed by Group Abort Behavior, for convenience.

Users and vendors can capture their own execution patterns by defining M1 behaviors to use as the type of compound behavioral connections. For example, some vendors might have an option on races to not abort the losing processes. This is a variation on the Race Behavior that does not have the finish-abort successions. It can be defined as an M1 instance of Compound Behavioral Connection that is always typed by the vendor-defined variant Race Behavior.

Course Event Conditions are Event Conditions for detecting Course Events, for example the start and ending of a behavior. It specifies the behavior producing the event with a behavior step, such as a step in a process or interaction in a choreography, and specifies the event with an event part, such as the parts for starting and ending (see the Happening Model). A Course Event condition can be the condition for an event monitoring step, enabling detection of the starting and ending of behaviors identified by behavior steps. For example, a Course Event condition can refer to a message part and the finish part in it to specify that the message has arrived (BPDM represents messages as special kinds of processes, see Simple Interaction Model).

Behavior Step Groups are Part Groups (see the Composition Model) that enclose Behavior Steps, and are also Behavior Steps themselves, typed by Behavior Occurrence in user models (M1). This gives a group of behavior steps as a whole the capacity to produce start and end events playing the standard event parts, such as start part and end part. For example, most process languages have a way of modeling sub processes without defining a separate process. This is a behavior step group.

6.2.2 Metamodel Specification

The **Behavior Model** enables **Behavior Steps** to be ordered in time as parts of other **Behavior Step** (see the **Course Model**). Vendors and users can define their own execution patterns with connections between these **Behavior Step**s. The model predefines a specific connection for races, where **Behavior Steps** start at the same time and abort each other when the first finishes. It also defines a **Behavior Event Condition** for detecting lifecycle events in **Behavior**. The **Behavior Model** is the most specialized model in the **Business Process Definition MetaModel** that still covers all of processes and interactions (orchestration and choreography, see the Activity and Interaction Protocol Models).

6.2.2.1 Behavior Model Diagram

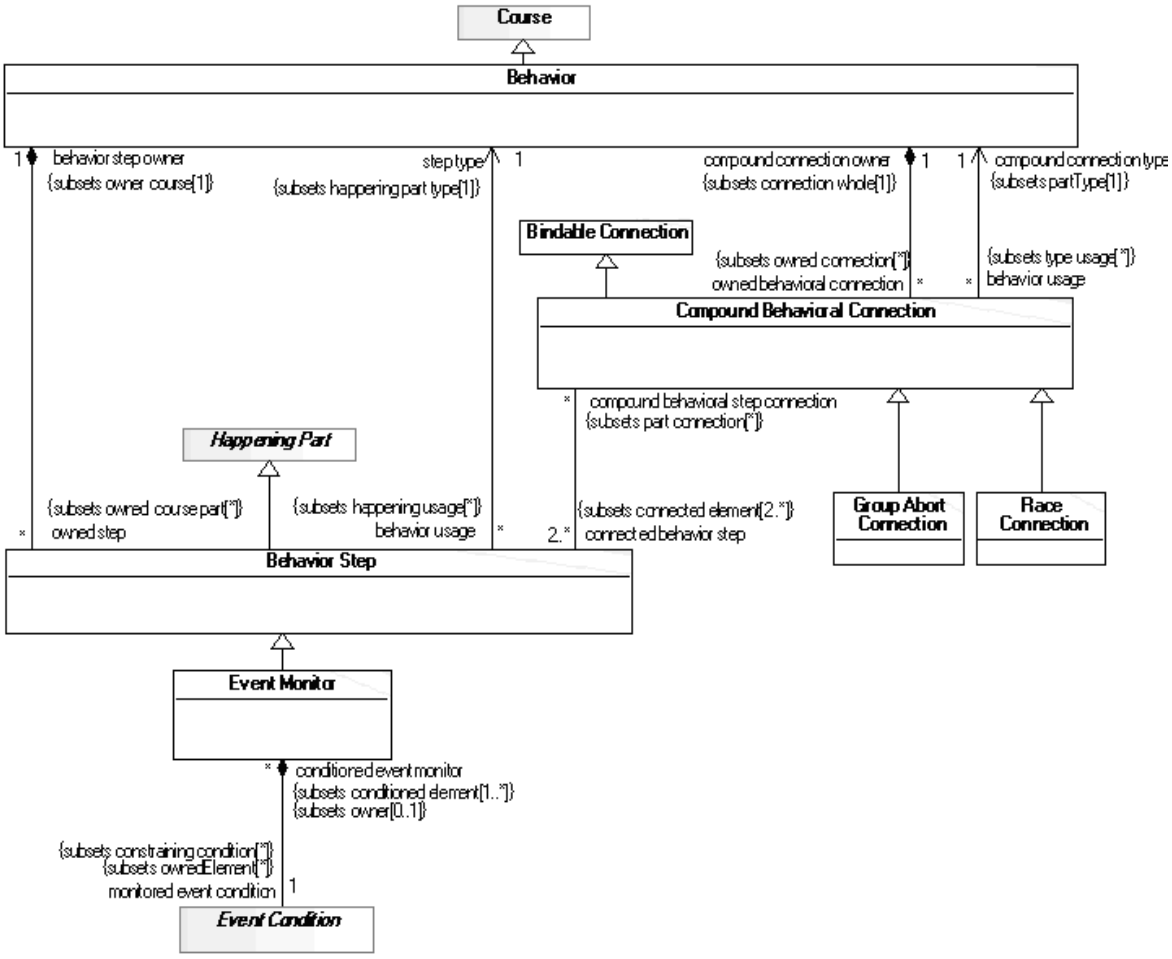


Figure 2 - Behavior Model Diagram

6.2.2.2 Behavior Library: Events

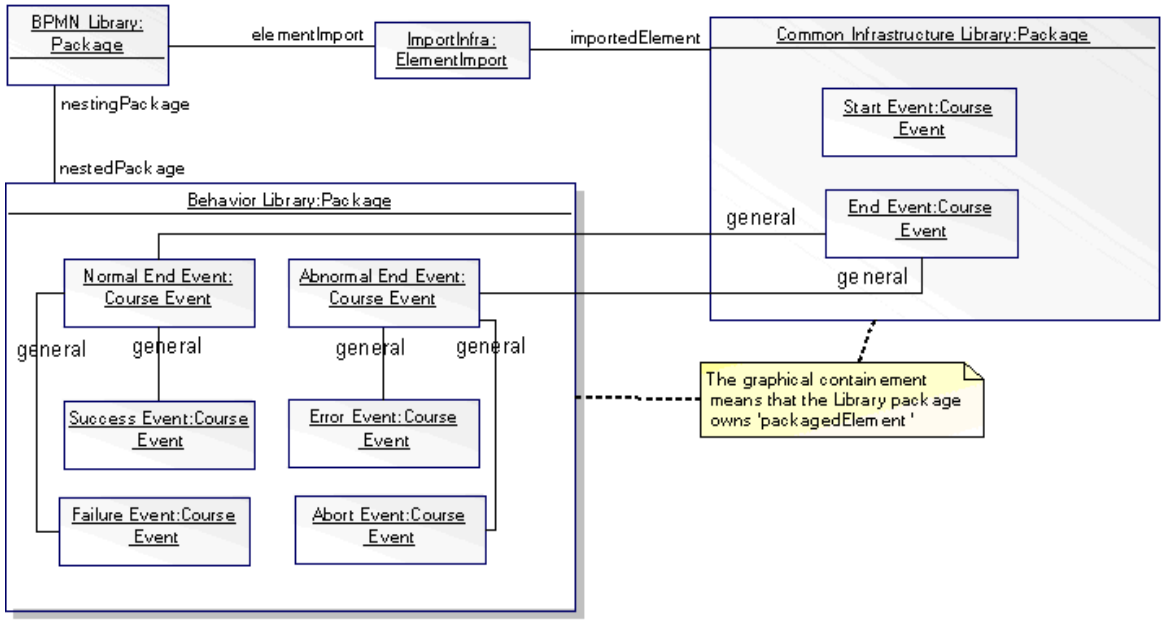


Figure 3 - Behavior Library: Events

6.2.2.3 Behavior Library: Behavior Occurrence

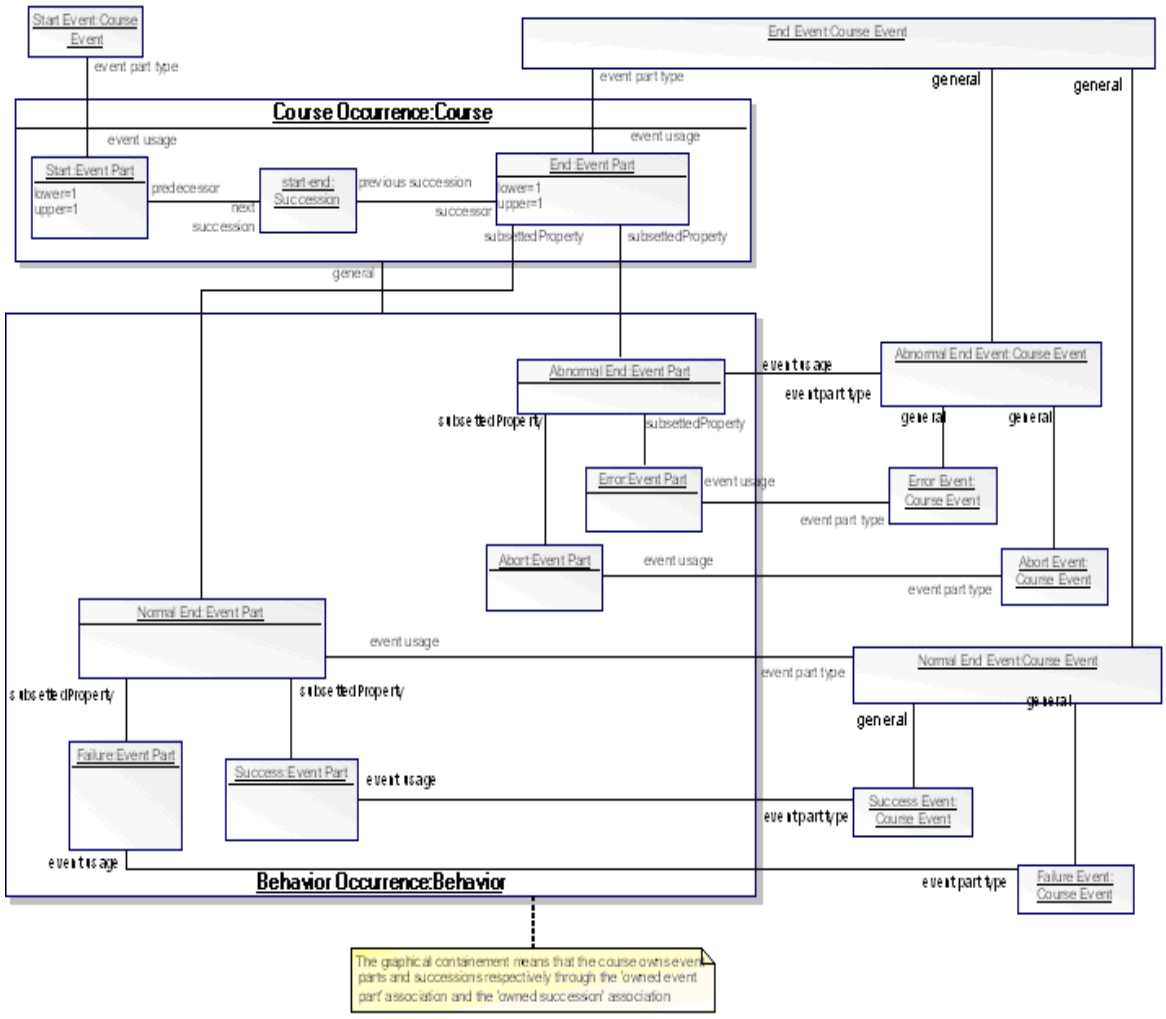


Figure 4 - Behavior Library : Behavior Occurrence

6.2.2.4 Behavior Library: 'Racing' Behavior

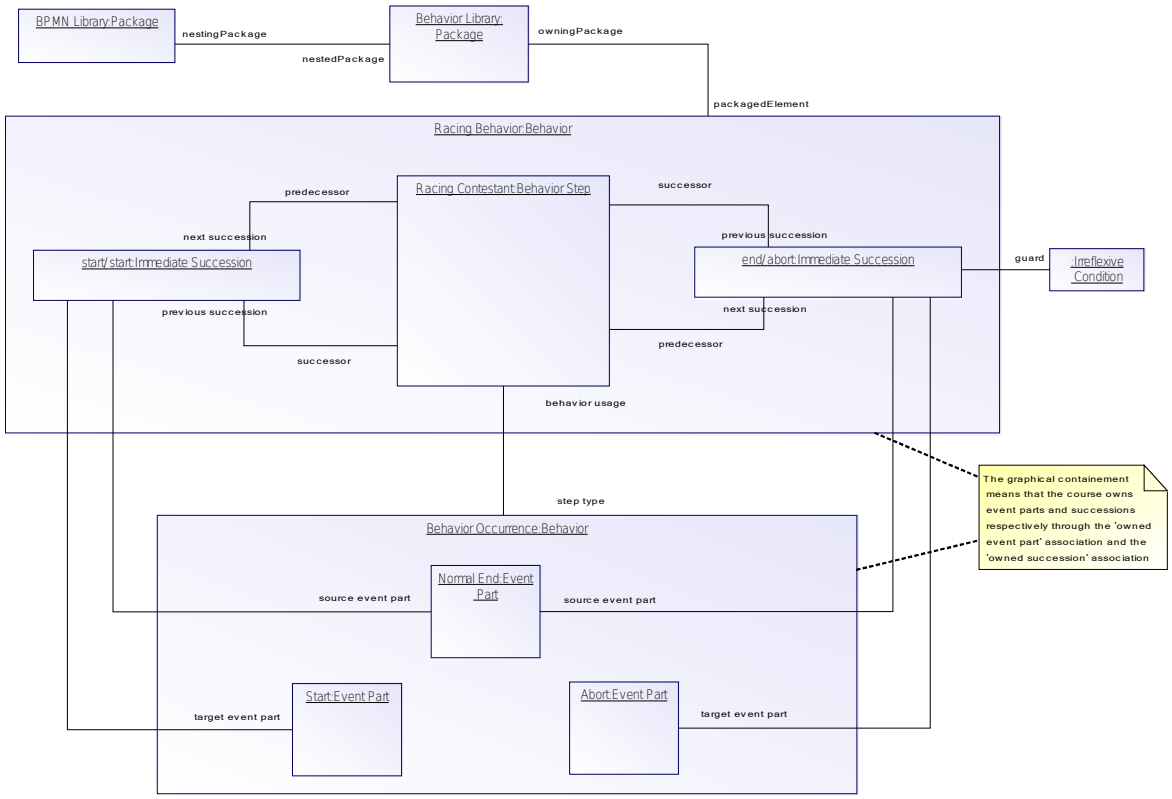


Figure 5 - Behavior Library: 'Racing' Behavior

6.2.2.5 Behavior Library: 'Group Abort Behavior'

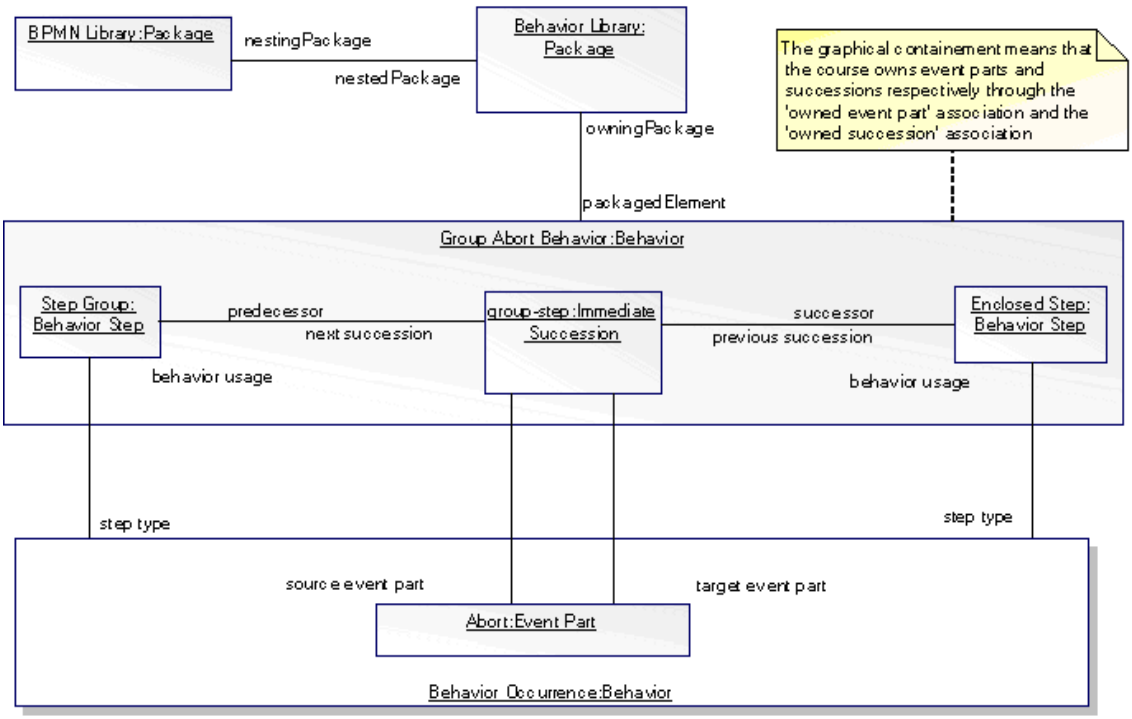


Figure 6 - Behavior Library: 'Group Abort Behavior'

6.2.2.6 Behavior Event Condition Diagram

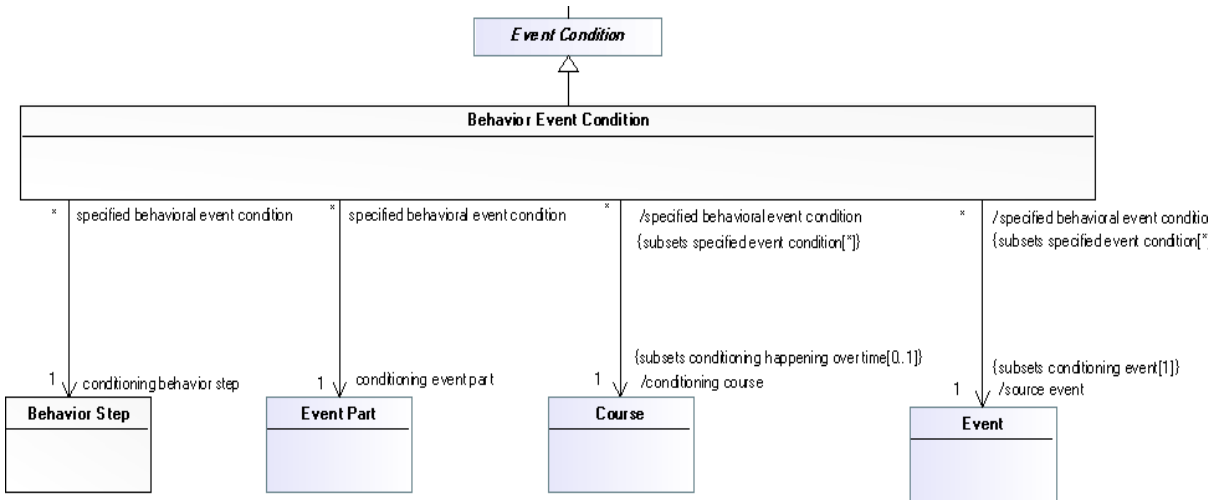


Figure 7 - Behavior Event Condition Diagram

6.2.2.7 Behavior Step Group Diagram

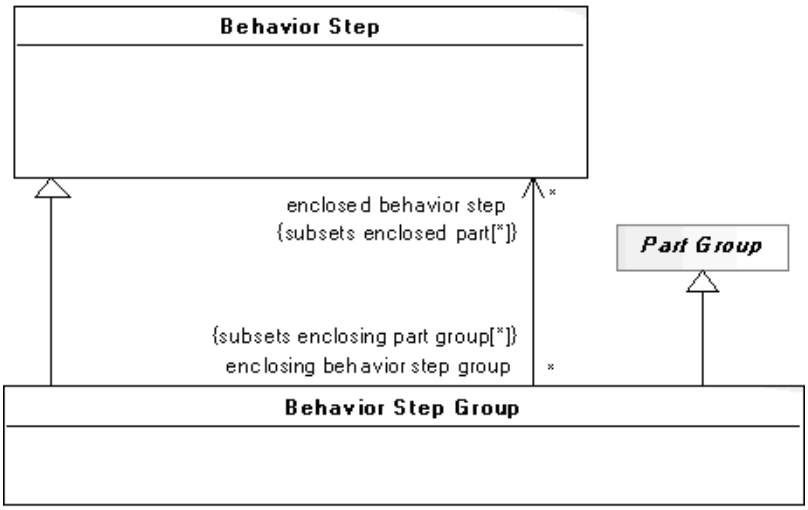


Figure 8 - Behavior Step Group Diagram

6.2.2.8 Connected Part Binding Diagram

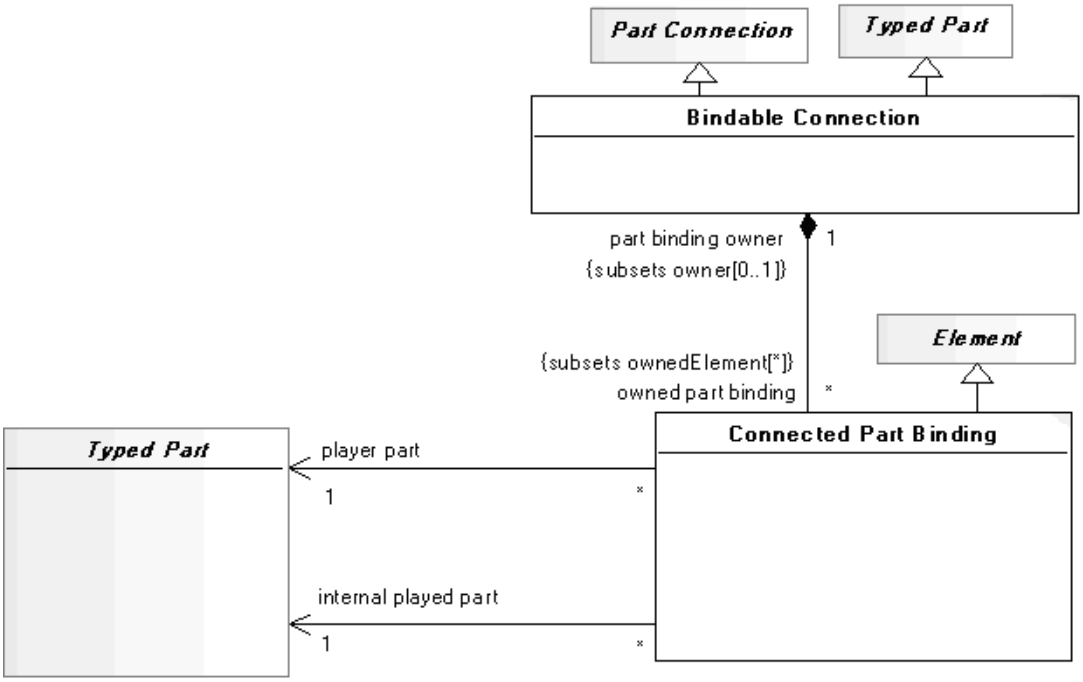


Figure 9 - Connected Part Binding Diagram

6.2.2.9 Behavior

Package: Behavior Model
isAbstract: No
Generalization: "Course"

Description

A **Behavior** is a kind of **Course** that order happenings in time, as in the activities of a process model and or the interactions in a choreography. **Behavior** introduces capabilities shared by both choreography and orchestration:

- Its steps are typed by **Course**s that provide them with start/end capabilities.
- As a **Course** it can organize its part with **Succession**. It adds the ability to order its steps according to their start and ends (**Succession**).
- Rich connections can be established between its steps to enable time sychronization between them (**Compound Behavioral Connection**).
- The reuse of the same **Behavior** is enabled by (**Behavior Step**).
- Detection of events in conditions, such as time events, fact changes, or behavior events can be to influence its course (**Event Monitor**).
- Its steps can be organized in groups to which start/end constraints can be applied (**Behavior Step Group**).

Associations

owned behavioral connection : Compound Behavioral Connection [*]	Compound Behavioral Connection owned by the Behavior Subsets *owned connection*
owned step : Behavior Step [*]	Behavior Step owned by the Behavior Subsets *owned course part*

6.2.2.10 Behavior Event Condition

Package: Behavior Model
isAbstract: No
Generalization: "Event Condition"

Description

Behavior Event Conditions are **Event Conditions** for detecting **Event**s in **Courses**, for example the start and ending of a **Course**.

It specifies the **conditioning behavior step**, such as a step in a process or interaction in choreography, and the **conditioning event part**, such as the **Event Part** for starting and ending (see the Happening and Event Model).

A **Behavior Event Condition** can be the condition for an **Event Monitor**, enabling detection of the starting and ending of **Course**s identified by behavior steps. For example, a **Behavior Event Condition** can refer to a message and the **Normal End** in it to specify that the message has arrived. (The **Business Process Definition MetaModel** represents messages as process steps themselves, see **Interactive Behavior Model**.)

Associations

conditioning behavior step : Behavior Step [1]	Behavior Step that is the source of the condition, such as an activity in a process or an interaction in a protocol.
conditioning course : Course [1]	Course that specifies the context of the Event that defines the condition. This is derived from conditioning behavior step of the condition. This is a derived association. Subsets *conditioning happening over time*
conditioning event part : Event Part [1]	Event Part that specifies the Event that is the source of the condition, such as the start (Start) or end (End).
source event : Event [1]	Event that specifies the Behavior Event Condition. This is derived from the Event Part that defines the Behavior Event Condition. This is a derived association. Subsets *conditioning event*

Constraint

[1] The **conditioning event part** must be an **Event Part** of the type of the **conditioning behavior step**.

6.2.2.11 Behavior Step

Package: Behavior Model
isAbstract: No
Generalization: "Happening Part"

Description

Behavior Steps is a kind of **Happening Part** which typed is a **Behavior**. This enables it to "invoke" other **Behavior** and to build **Behavior** composites (made of sub- **Behaviors**).

Associations

compound behavioral step connection : Compound Behavioral Connection [*]	Compound Behavioral Connection indicating that the lifecycle of the Behavior Step is tied to the life cycle of other Behavior Steps. Subsets *part connection*

step type : Behavior [1]	Specifies the type of the Behavior Step. The default step type is the Behavior Occurrence. Subsets *happening part type* Default: Behavior Occurrence

6.2.2.12 Behavior Step Group

Package: Behavior Model
isAbstract: No
Generalization: "Behavior Step" "Part Group"

Description

A **Behavior Step Group** is a kind of **Part Group** that is also a **Behavior Step** typed by the **Behavior Occurrence** in user models (M1). This gives a group of **Behavior Steps** as a whole the capacity to produce start and end changes playing the standard Event Parts, such as **Start** and **End**. For example, most process languages have a way of modeling sub-processes without defining a separate process. This is a **Behavior Step Group**.

Associations

enclosed behavior step : Behavior Step [*]	Behavior Step being part of the Behavior Step Group Subsets *enclosed part*

6.2.2.13 Bindable Connection

Package: Behavior Model
isAbstract: No
Generalization: "Part Connection" "Typed Part"

Description

A **Bindable Connection** is a kind of **Part Connection** defined just to categorize those connections that can carry **Connected Part Binding**.

Associations

owned part binding : Connected Part Binding [*]	Connected Part Binding owned by the Composite. Subsets *ownedElement*

6.2.2.14 Compound Behavioral Connection

Package: Behavior Model
isAbstract: No
Generalization: "Bindable Connection"

Description

A **Compound Behavioral Connection** is a **Part Connection** that enables dedicated lifecycle rule connections to apply between **Behavior Steps**. These rules are described by the **compound connection type** of the **Compound Behavioral Connection**, which is itself a **Behavior**. This makes **Compound Behavioral Connection** be itself a **Typed Part**.

Associations

compound connection type : Behavior [1]	Behavior typing the Compound Behavioral Connection and specifying the lifecycle rules (start/start, abort/abort) tying all Behavior Steps connected by the Compound Behavioral Connection. Subsets *partType*
connected behavior step : Behavior Step [2..*]	Behavior Step connected by the Compound Behavioral Connection. Subsets *connected element*

6.2.2.15 Connected Part Binding

Package: Behavior Model
isAbstract: No
Generalization: "Element"

Description

A **Connected Part Binding** is an **Element** specifying that individuals playing the part at an end of a **Part Connection** also play a **Part** within the connection. For example, one of the interactions between businesses in a choreography might be a subchoreography composed of many communications between the businesses. Businesses playing a particular role in the larger choreography also play one of the roles in the subchoreography.

The player is part of the composite owning the bindable connection. The played is part of the bindable connection. The binding requires the (M0) individuals playing these parts to be the same. They are found by navigating from an individual composite, to the player individuals, and to the played individuals in the connection part of the same composite. The two sets of individuals found this way must be exactly the same.

Connected Part Binding is different from **Part Connection** because part bindings are about which individuals are playing certain parts in a whole, whereas connections are about links between the individuals themselves due to playing parts in the whole. As a convenience, it is assumed that a connection typed by a composite that has only one (non-connection) part implies bindings where that one part is played by all the parts at all the ends of the connector. This is useful for symmetrical connectors.

Associations

internal played part : Typed Part [1]	The played is part of the bindable connection.
player part : Typed Part [1]	The player is part of the composite owning the bindable connection.

6.2.2.16 Event Monitor

Package: Behavior Model
isAbstract: No
Generalization: "Behavior Step"

Description

An **Event Monitor** is a kind of **Behavior Step** that monitors the occurrence of an **Event Condition** and that has an effect on the course of a **Behavior**. For instance, an **Event Monitor** can be used to react to the **Abort Event** of a specific **Course**.

Associations

monitored event condition : Event Condition [1]	Event Condition being monitored. Subsets *constraining condition* Subsets *ownedElement*

BPMN Notation

Event Monitor shape with the marker of the **Compensate Event** instance of **Event**.

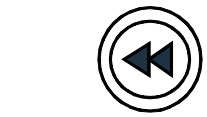


Figure 10 - Event Monitor monitoring a 'Compensate' Course Event Condition

Figure 11 - Event Monitor monitoring a Compound Event Condition

Event Monitor shape with a **Fact Change** as a maker.

Figure 12 - Event Monitor monitoring a Fact Change Condition

Event Monitor shape with a **Time Event** as a maker.

Figure 13 - Event Monitor monitoring a Time Event Condition

This symbol is a circle, with an open center. The circle MUST be drawn with a double thin black line. It can alternatively represent:

1. **Event Parts** that are not typed by **Start Event** or **End Event**.
2. **Event Monitors**

Markers can be placed within the circle to indicate the nature of the **Event** associated with the **Event Part** or **Event Monitor**.

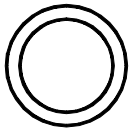


Figure 14 - Event Monitor Notation

6.2.2.17 Group Abort Connection

Package: Behavior Model
isAbstract: No
Generalization: "Compound Behavioral Connection"

Description

A **Group Abort Connection** is a kind of **Compound Behavioral Connection** that has for **compound connection type** the **Group Abort Behavior**, an M1 instance of **Behavior** defined in the **Behavior Library** user (M1) library. It is applied to **Behavior Step Groups** and their enclosed steps to ensure that the steps are aborted when the group is. (See more details in **Group Abort Behavior**).

6.2.2.18 ImmediateSuccession

Package: Behavior Model
isAbstract:
Generalization: "Immediate Succession"

Description

Immediate Succession in the **Business Process Definition MetaModel** namespace.

6.2.2.19 Race Connection

Package: Behavior Model
isAbstract: No
Generalization: "Compound Behavioral Connection"

Description

A **Race Connection** is a kind of **Compound Behavioral Connection** that has for **compound connection type** the **Racing Behavior**. The **Racing Behavior** ensures that all the connected **Behavior Steps** start at the same time, and that the first one to finish aborts the others.

6.2.2.20 Succession

Package: Behavior Model
isAbstract:
Generalization: "Succession"

Description

Succession in the **Business Process Definition MetaModel** namespace.

6.2.2.21 Instance: Abnormal End Event

Class: Course Event

Description

Abnormal End Event is an **Event** that manifests the abnormal **End Event** of a **BehaCourse**.

Links

Played End	*Opposite End*
Abnormal End Event:	*general* End Event
Abnormal End Event:event part type	*event usage* Abnormal End
Abnormal End Event:general	Abort Event
Abnormal End Event:general	Error Event
Abnormal End Event:packagedElement	*owningPackage* Behavior Library

Non Normative Notation

Marker of the **Normal End** instance of **Event**.

'Abnormal End' Behavioral Event Instance

Figure 15 - Course Event 'Abnormal End' instance notation

6.2.2.22 Instance: Abnormal End

Class: Event Part

Description

Links

Played End	*Opposite End*
Abnormal End:	*subsettedProperty* End
Abnormal End:event usage	*event part type* Abnormal End Event
Abnormal End:owned event part	*event part owner* Behavior Occurrence
Abnormal End:subsettedProperty	Error
Abnormal End:subsettedProperty	Abort

Non Normative Notation

The shape of the **Abnormal End** instance uses the shape of its super-property (**End**) with marker of its type: **Abnormal End Event**.

'Abnormal End' Event Part

Figure 16 - Event Part : Abnormal End notation

6.2.2.23 Instance: Abort Event

Class: Course Event

Description

Abort Event is an **Event** that manifests that the course of a **Course** is being interrupted. The source of the **Abort Event** can be internal or external to the **Course**.

Links

Played End	*Opposite End*
Abort Event:	*general* Abnormal End Event
Abort Event:event part type	*event usage* Abort
Abort Event:induced course event	*course event context* Abort Process
Abort Event:packagedElement	*owningPackage* Behavior Library

BPMN Notation

Marker of the **Abort Event** instance of **Event**.

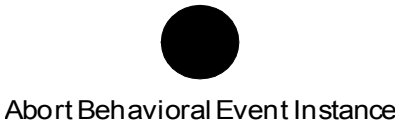


Figure 17 - Course Event 'Abort' Notation

6.2.2.24 Instance: Abort

Class: Event Part

Description

Links

Played End	*Opposite End*
Abort:	*subsettedProperty* Abnormal End
Abort:event usage	*event part type* Abort Event
Abort:owned event part	*event part owner* Behavior Occurrence
Abort:source event part	group-step
Abort:target event part	end/abort
Abort:target event part	group-step

BPMN Notation

The shape of the **Abort** instance of **Event Part** uses the shape of its super-property (**End**) with the marker of its event type: **Abort Event**.

Figure 18 - Event Part : Abort Notation

6.2.2.25 Instance: Behavior Library

Class: Package

Description

User (M1) library capturing commonly needed aspects of happenings as instances of the class in the **Happening Model** model. The library defines:

- **Events** to represent various behavior lifecycle events, such as starting and ending of individual **Courses**.
- A **Course** is called the **Behavior Occurrence**. It is a generalization of all M1 dynamic models (see the **Composition Model**).
- **Event Parts** of the **Behavior Occurrence** for the various Events, such as start and end. These are typed by the various M1 changes, such as Start and End Events.

Successions between the **Event Parts** above for universal constraints, such as the End being after the Start.

Links

Played End	*Opposite End*
Behavior Library:nestedPackage	*nestingPackage* BPMN Library
Behavior Library:owningPackage	*packagedElement* Abnormal End Event
Behavior Library:owningPackage	*packagedElement* Normal End Event
Behavior Library:owningPackage	*packagedElement* Error Event
Behavior Library:owningPackage	*packagedElement* Abort Event
Behavior Library:owningPackage	*packagedElement* Behavior Occurrence
Behavior Library:owningPackage	*packagedElement* Success Event
Behavior Library:owningPackage	*packagedElement* Failure Event

6.2.2.26 Instance: Behavior Library

Class: Package

Description

Package including the standard **Group Abort Behavior** and **Racing Behavior**.

Links

Played End	*Opposite End*
Behavior Library:nestedPackage	*nestingPackage* BPMN Library
Behavior Library:owningPackage	*packagedElement* Racing Behavior
Behavior Library:owningPackage	*packagedElement* Group Abort Behavior

6.2.2.27 Instance: Behavior Occurrence

Class: Behavior

Description

Course that produces common behavior lifecycle changes, such as **Start** or **End Event**.

Links

Played End	*Opposite End*
Behavior Occurrence:	*general* Course Occurrence
Behavior Occurrence:event part owner	*owned event part* Error
Behavior Occurrence:event part owner	*owned event part* Normal End
Behavior Occurrence:event part owner	*owned event part* Abort
Behavior Occurrence:event part owner	*owned event part* Success
Behavior Occurrence:event part owner	*owned event part* Failure
Behavior Occurrence:event part owner	*owned event part* Abnormal End
Behavior Occurrence:general	Generalization
Behavior Occurrence:general	Generalization
Behavior Occurrence:packagedElement	*owningPackage* Behavior Library
Behavior Occurrence:step type	*behavior usage* Activity 1
Behavior Occurrence:step type	*behavior usage* Racing Contestant
Behavior Occurrence:step type	*behavior usage* Step Group
Behavior Occurrence:step type	*behavior usage* Enclosed Step

Constraint

[1] Normal End and Abnormal End cannot have values at the same time.
not (self.Normal End->notEmpty() and self.Abnormal End->notEmpty())

[2] Failure and Success cannot have values at the same time.
not (self.Failure->notEmpty() and self.Success->notEmpty())

[3] Abort and Error cannot have values at the same time.
not (self.Abort->notEmpty() and self.Error->notEmpty())

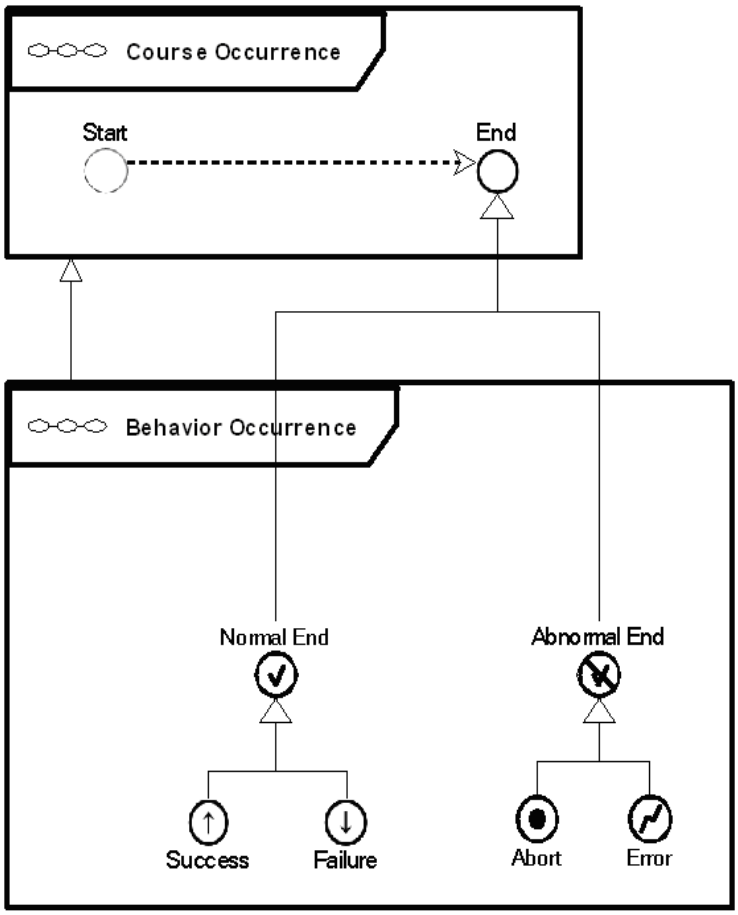


Figure 19 - Behavior Occurrence

6.2.2.28 Instance: Enclosed Step

Class: Behavior Step

Description

Represents the behavior of a **Behavior Step** within a **Behavior Step Group**.

Links

Played End	*Opposite End*
Enclosed Step:behavior usage	*step type* Behavior Occurrence
Enclosed Step:owned step	*behavior step owner* Group Abort Behavior
Enclosed Step:successor	*previous succession* group-step

6.2.2.29 Instance: end/abort

Class: Immediate Succession

Description

This succession has the finish part on one end and the abort part on the other, specifying that any contestant

happening that finishes will be accompanied by a simultaneous abort of the others. This succession has the Irreflexive condition applied (see the Composition Model), to prevent the finishing contestant from aborting itself.

Links

Played End	*Opposite End*
end/abort:	*target event part* Abort
end/abort:	*source event part* Normal End
end/abort:	*guard* Irreflexive Condition
end/abort:next succession	*predecessor* Racing Contestant
end/abort:owned succession	*owner course* Racing Behavior
end/abort:previous succession	*successor* Racing Contestant

6.2.2.30 Instance: Error Event

Class: Course Event

Description

Error Event is an **Event** that manifests that an error has occurred that will lead to the **End Event** of the **Course**. The source of the **Error Event** is always internal to the **Course**.

Links

Played End	*Opposite End*
Error Event:	*general* Abnormal End Event
Error Event:event part type	*event usage* Error
Error Event:induced course event	*course event context* Error Process
Error Event:packagedElement	*owningPackage* Behavior Library

BPMN Notation

Marker of the **Error Event** instance of **Event**.

Figure 20 - Course Event 'Error' Instance Notation

6.2.2.31 Instance: Error

Class: Event Part

Description

Links

Played End	*Opposite End*
Error:	*subsettedProperty* Abnormal End
Error:event usage	*event part type* Error Event
Error:owned event part	*event part owner* Behavior Occurrence
Error:source event part	error handling

BPMN Notation

This symbol can alternatively represent:

1. **Event Part** typed by the **Error Event** instance of **Event**.
2. An **Error Activity**

Figure 21 - Error Activity Notation or 'Error' Course Event Step

Error Event Event Part used for error handling. The **Error Event Event Part** is linked to the **Succession** instance through the **source event part** association.

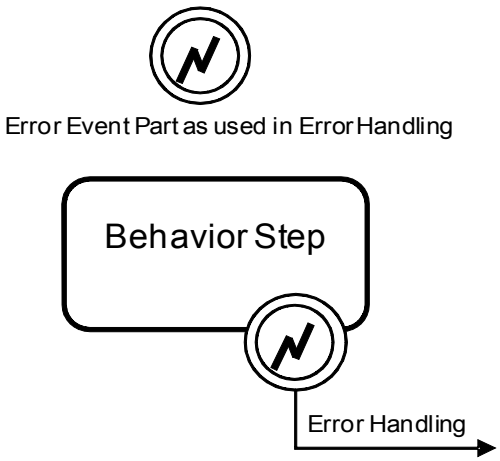


Figure 22 - Error Handling Notation

The shape of the **Error** instance of **Event Part** uses the shape of its super-property (**End**) with the marker of its event type: **Error Event**.

Figure 23 - Event Part : Error Notation

6.2.2.32 Instance: Failure Event

Class: Course Event

Description

Failure Event is a kind of **End Event** that indicates that its **Course** has ended, but has not reached its purpose.

Links

Played End	*Opposite End*
Failure Event:	*general* Normal End Event
Failure Event:event part type	*event usage* Failure
Failure Event:packagedElement	*owningPackage* Behavior Library

Non-normative Notation

Marker of the **Failure Event** instance of **Event**.

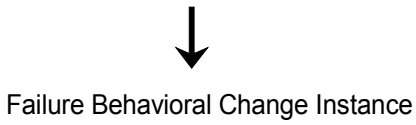


Figure 24 - Course Event 'Failure' Instance notation

6.2.2.33 Instance: Failure

Class: Event Part

Description

Links

Played End	*Opposite End*
Failure:	*subsettedProperty* Normal End
Failure:event usage	*event part type* Failure Event
Failure:owned event part	*event part owner* Behavior Occurrence

Non Normative Notation

The shape of the **Failure** instance uses the shape of its super-property (**End**) with marker of its type:**Failure Event**.

Figure 25 - Event Part : Failure Notation

6.2.2.34 Instance: Group Abort Behavior

Class: Behavior

Description

Group Abort Behavior contains:

- Two steps, one for the group and one for its enclosed steps (Step Group and Enclosed Step). The first is bound to an M1 processing step group and the second to each step in the group (see Connected Part Binding above).

- One immediate processing succession between the two steps above. The source is Step Group and the target is Enclosed Step. It refers to the abort part on both ends (see the Happening and Change Model), specifying that any group behavior that aborts will be accompanied by a simultaneous abort of the enclosed step happenings.

- When a group abort connection is created between a processing step group and its steps, it implies a part binding between Step Group in the Group Abort Behavior and the connected group, with Step Group on the played end (see Connected Part Binding above). Similarly, it implies bindings between Enclosed Step and the steps in the group. The part bindings ensure that any individual M0 happening playing the connected group will also play the Step Group, and any individual playing the connected steps will also play the Enclosed Step, establishing the abort-abort successions between the connected group and steps, and the temporal constraints on the individual happenings. The Group Abort Behavior above can be the type for any connector that is also a typed part, but Group Abort Connection is always typed by Group Abort Behavior, for convenience.

Links

Played End	*Opposite End*
Group Abort Behavior:behavior step owner	*owned step* Enclosed Step
Group Abort Behavior:behavior step owner	*owned step* Step Group
Group Abort Behavior:owner course	*owned succession* group-step
Group Abort Behavior:packagedElement	*owningPackage* Behavior Library

6.2.2.35 Instance: group-step

Class: Immediate Succession

Description

Links

Played End	*Opposite End*
group-step:	*source event part* Abort
group-step:	*target event part* Abort
group-step:next succession	*predecessor* Step Group
group-step:owned succession	*owner course* Group Abort Behavior
group-step:previous succession	*successor* Enclosed Step

6.2.2.36 Instance: ImportInfra

Class: ElementImport

Description

Import of the **Common Infrastructure Library**

Links

Played End	*Opposite End*
ImportInfra:	*importedElement* Common Infrastructure Library
ImportInfra:elementImport	BPMN Library

6.2.2.37 Instance: Normal End Event

Class: Course Event

Description

Normal End Event is an **Event** that manifests the normal **End Event** of a **Course**.

Links

Played End	*Opposite End*
Normal End Event:	*general* End Event
Normal End Event:event part type	*event usage* Normal End
Normal End Event:general	Success Event
Normal End Event:general	Failure Event
Normal End Event:packagedElement	*owningPackage* Behavior Library

Non-normative Notation

Marker of the **Normal End Event** instance of **Event**.

'Normal End' Behavioral Event Instance

Figure 26 - Course Event 'Normal End' instance notation

6.2.2.38 Instance: Normal End

Class: Event Part

Description

Played End	*Opposite End*
Normal End:	*subsettedProperty* End
Normal End:event usage	*event part type* Normal End Event
Normal End:owned event part	*event part owner* Behavior Occurrence
Normal End:source event part	end/abort
Normal End:source event part	start/start
Normal End:subsettedProperty	Success
Normal End:subsettedProperty	Failure

Non Normative Notation

The shape of the **Normal End** instance uses the shape of its super-property (**End**) with the marker of its type: **Normal End Event**.

'Normal End' Event Part

Figure 27 - Event Part : Normal End notation

6.2.2.39 Instance: Racing Behavior

Class: Behavior

Description

Racing Behavior contains:

- One **Behavior Step**, called the **Racing Contestant**, which is bound to all the steps connected by the M1 race connection. This ensures that all the contestants are treated the same way.

- Two **Immediate Processing Succession**s connecting the Contestant to itself. One succession refers to the start part of the Contestant on both ends (see the Happening and Change Model), specifying that all the contestant behaviors start at the same time. The other succession has the finish part on one end and the abort part on the other, specifying that any contestant happening that finishes will be accompanied by a simultaneous abort of the others. This succession has the Irreflexive condition applied (see the Composition Model), to prevent the finishing contestant from aborting itself.

Links

Played End	*Opposite End*
Racing Behavior:behavior step owner	*owned step* Racing Contestant
Racing Behavior:owner course	*owned succession* start/start
Racing Behavior:owner course	*owned succession* end/abort
Racing Behavior:packagedElement	*owningPackage* Behavior Library

6.2.2.40 Instance: Racing Contestant

Class: Behavior Step

Description

Behavior Step of the **Racing Behavior** is bound to all the steps connected by the M1 race connection to ensure that all the contestants are treated the same way.

Links

Played End	*Opposite End*
Racing Contestant:behavior usage	*step type* Behavior Occurrence
Racing Contestant:owned step	*behavior step owner* Racing Behavior
Racing Contestant:predecessor	*next succession* start/start
Racing Contestant:predecessor	*next succession* end/abort
Racing Contestant:successor	*previous succession* start/start
Racing Contestant:successor	*previous succession* end/abort

6.2.2.41 Instance: start/start

Class: Immediate Succession

Description

This succession refers to the start part of the **Racing Contestant** on both ends (see the Happening and Change Model introduction), specifying that all the contestant behavior start at the same time.

Links

Played End	*Opposite End*
start/start:	*source event part* Normal End
start/start:	*target event part* Start

Played End	*Opposite End*
start/start:next succession	*predecessor* Racing Contestant
start/start:owned succession	*owner course* Racing Behavior
start/start:previous succession	*successor* Racing Contestant

6.2.2.42 Instance: Step Group

Class: Behavior Step

Description

Represents the behavior of a **Behavior Step Group** regarding its **Enclosed Step**.

Links

Played End	*Opposite End*
Step Group:behavior usage	*step type* Behavior Occurrence
Step Group:owned step	*behavior step owner* Group Abort Behavior
Step Group:predecessor	*next succession* group-step

6.2.2.43 Instance: Success Event

Class: Course Event

Description

Success Event is a kind of **End Event** that indicates that its **Course** has ended by fulfilling its purpose.

Links

Played End	*Opposite End*
Success Event:	*general* Normal End Event
Success Event:event part type	*event usage* Success
Success Event:packagedElement	*owningPackage* Behavior Library

Non Normative Notation

Marker of the **Success Event** instance of **Event**.

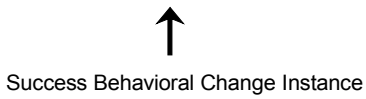


Figure 28 - Course Event 'Success' Instance notation

6.2.2.44 Instance: Success

Class: Event Part

Description

Links

Played End	*Opposite End*
Success:	*subsettedProperty* Normal End
Success:event usage	*event part type* Success Event
Success:owned event part	*event part owner* Behavior Occurrence

Non Normative Notation

The shape of the **Success** instance uses the shape of its super-property (**End**) with marker of its type: **Success Event**.

Figure 29 - Event Part : Success Notation

6.3 Interactive Behavior Model

6.3.1 Introduction

The Interactive Behavior Model enables interactions to be treated like any other step in a behavior, ordered in time, with start and end events. The model is the basis for flows between process steps and between participants in a choreography (see the Activity Model and the Interaction Protocol Model). The Interactive Behavior Model is the most specialized model in BPDM that still has elements in common between orchestration and choreography.

The Interactive Behavior Model provides:

- Behaviors with interactions and roles (Interactive Behavior)
- Interactions that have no sub interactions (Simple Interactions).
- Types for flowing entities (transferred item types).
- Expressions for changing which entities are flowing (transformation expression).
- Parts that interact within a behavior (Interaction Roles).

Interactive Behaviors are Behaviors that can have interactions as parts. Interactions are Typed Part Connections that are also Behavior Steps, enabling them to have start and end events, and be ordered in time. This is used to define reusable protocols and specify the way a process interacts with its environment (see the Interaction Protocol Model and the Activity Model). Interactive Parts are defined just to categorize those Typed Parts that can be connected by Interactions. The types of interactive parts establish requirements for the interacting individuals, for example, that they have a minimum security clearance or market capitalization.

Simple Interactions are interactions in which something is "transferred" from individuals playing one interactive part to individuals playing another interactive part. For example, a document, phone number, or package may be transferred from one department to another in a company. The transferred items must conform to a Type specified by the simple interaction. Simple Interactions can have an expression to change the item that arrives at the target based on the item flowing from the source. For example, a transformation may retrieve the zip code from an address flowing from the source to deliver the zip code to the target.

Simple Interactions in user (M1) models are always typed by the Behavior Occurrence (see user library in the Behavior Model). This gives them the standard event parts, such as for start and end, so the simple interactions can be ordered within an Interaction Protocol (see the Interaction Protocol Model). This is different from the type of thing transferred.

Simple interactions can be bound to each other for specifying that a simple interaction is the same as some of the simple interactions in the interactive parts it connects. For example, an interaction between steps in a process can be bound to interactions in the connected steps that output and input transferred items (see the Activity Model). The individuals constrained by binding are interactions as they occur at M0, for example, transferring a car with a certain

identification number at a certain time. These individual (M0) interactions are found by navigating from an individual composite, to individual interactions playing a part in it, and from there to internal interactions in the source end, and to internal interactions in the target end. The three sets of individuals found this way must be exactly the same. Simple interaction binding is different from connections because interaction binding is about which individuals are playing certain parts in a whole, whereas connections are about links between the individuals themselves due to playing parts in the whole.

Interaction Roles are Interactive Parts played by individuals outside the behavior, but interacting with it. For example, the customer is an interaction role in a behavior for delivering a product. This is specialized in other BPDM packages for application to orchestration and choreography (see the Activity Model and the Interaction Protocol Model).

6.3.2 Metamodel Specification

The **Interactive Behavior Model** enables interactions to be treated like any other step in a **Behavior**, ordered in time, with start and end events. The model is the basis for flows between **Behavior Step**s and between participants in a choreography (see the Activity Model and the Interaction Protocol Model). The **Interactive Behavior Model** is the most specialized model in the **Business Process Definition MetaModel** that still has elements in common between processes and choreographies.

6.3.2.1 Interactive Behavior Diagram

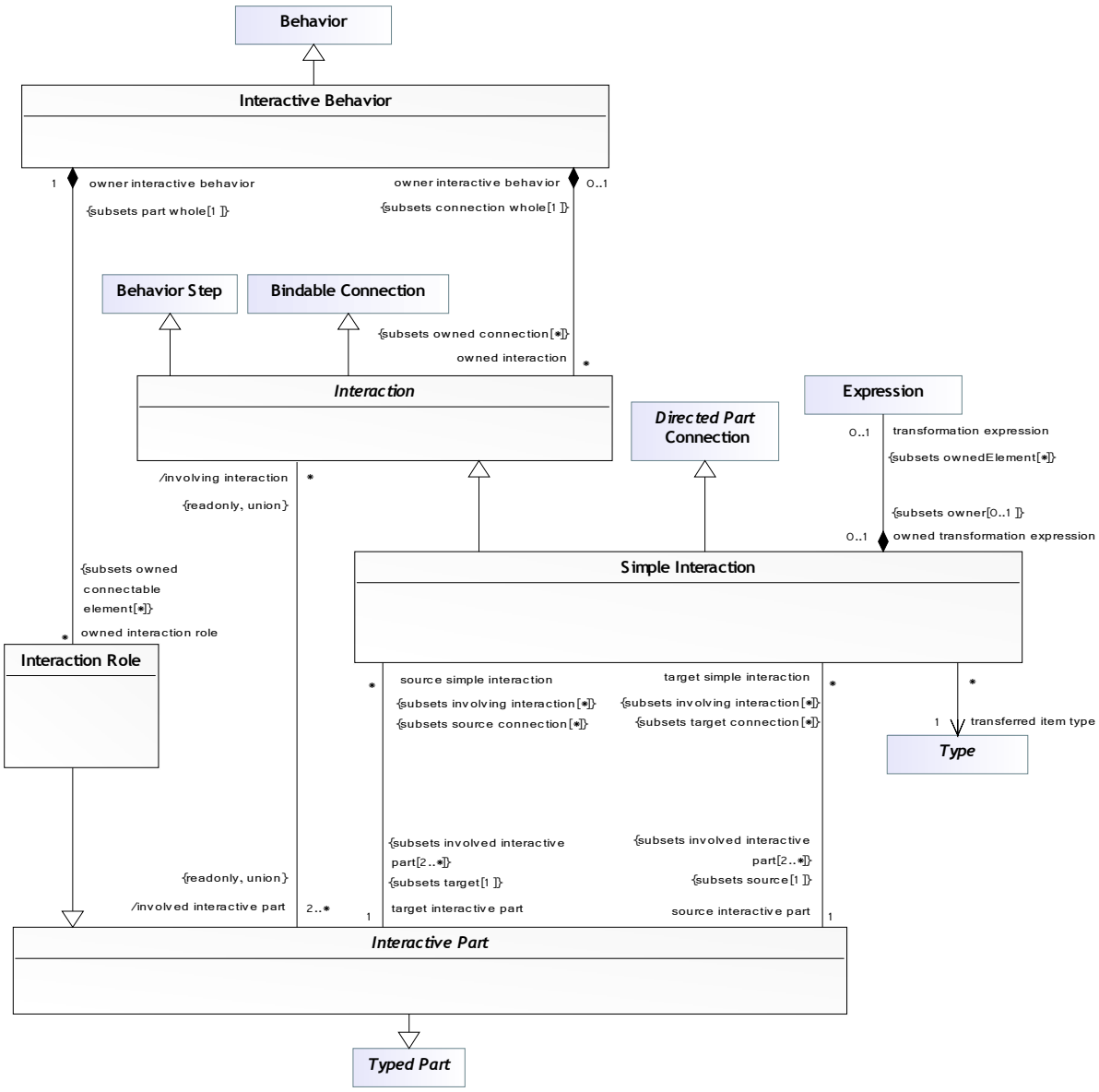


Figure 30 - Interactive Behavior Diagram

6.3.2.2 Simple Interaction Binding Diagram

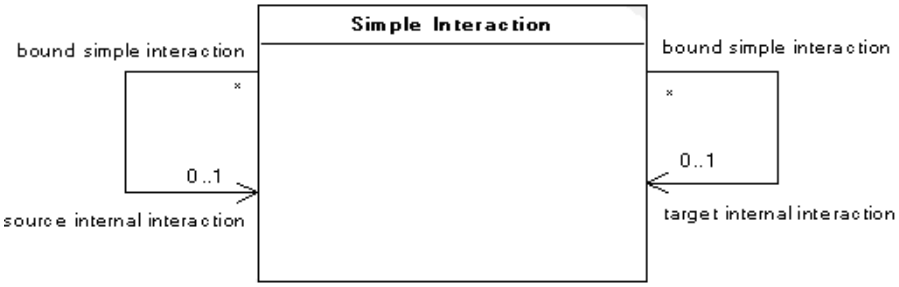


Figure 31 - Simple Interaction Binding Diagram

6.3.2.3 Message Diagram

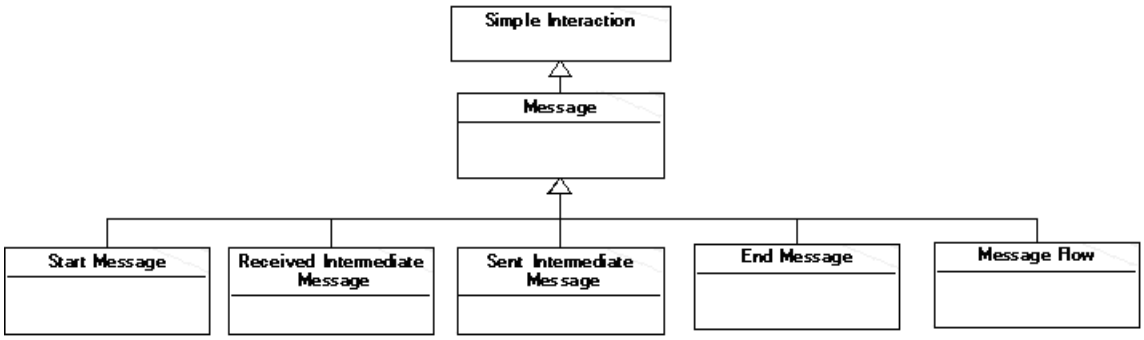


Figure 32 - Message Diagram

6.3.2.4 End Message

Package: Interactive Behavior Model
isAbstract: No
Generalization: "Message"

Description

An **End Message** is a **Message** that has the following additional characteristics:

- Its **target interactive part** is an **Interaction Role**.
- It has a **previous succession**.
- It has a **next succession** to the **End** instance of **Event Part**.
- This **Succession** is an **Immediate Succession**.

The sending the message is simultaneous with the end of the process.

BPMN Notation

Notation for **End Message** or **Simple Interaction** categorized as an **End Message**.

Figure 33 - End Message Notation

6.3.2.5 Interaction

Package: Interactive Behavior Model
isAbstract: Yes
Generalization: “Behavior Step” “Bindable Connection”

Description

An **Interaction** is a **Behavior Step** that is also a **Part Connection** , enabling **Interaction** to have start and end changes, and be ordered in time.

An **Interaction** can be either a simple **Simple Interaction** or a set of combined **Simple Interaction**s : a **Compound Interaction**. Ultimately, an **Interaction** is realized by the exchange of **Simple Interaction**s between its **Interactive Part**s.

Associations

involved interactive part : Interactive Part [2..*]	Interactive Part involved in the Interaction. This is a derived union.

6.3.2.6 Interaction Role

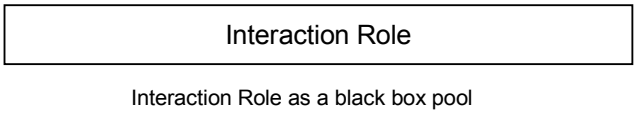
Package: Interactive Behavior Model
isAbstract: No
Generalization: “Interactive Part”

Description

An **Interaction Role** is an **Interactive Part** where the individuals playing the part are in the environment context where the **Behavior** is used. For example, the customer is an **Interaction Role** in a behavior for delivering a product.

BPMN Notation

A "black box pool" is a pool that does not have any process details.



Interaction Role as a black box pool

Figure 34 - Interaction Role Notation

6.3.2.7 Interactive Behavior

Package: Interactive Behavior Model
isAbstract: No
Generalization: "Behavior"

Description

An **Interactive Behavior** is a kind of **Behavior** that can have **Interaction**s as its **Part**s. To be involved in **Interaction**s, these **Part**s must be sub-types of **Interactive Part**.

Associations

owned interaction role : Interaction Role [*]	Interaction Role owned by the Interactive Behavior. Subsets *owned connectable element*
owned interaction : Interaction [*]	Interaction owned by the Behavior Subsets *owned connection*

6.3.2.8 Interactive Part

Package: Interactive Behavior Model
isAbstract: Yes
Generalization: "Typed Part"

Description

Interactive Part is a category of **Typed Part** that can be connected by **Interactions**. The types of interactive parts establish requirements for the interacting individuals, for example, that they have a minimum security clearance or market capitalization.

Associations

involving interaction : Interaction [*]	Interaction that the Interactive Part is involved in. This is a derived union.
source simple interaction : Simple Interaction [*]	Simple Interaction going to the target interactive part. Subsets *involving interaction* Subsets *source connection*
target simple interaction : Simple Interaction [*]	Simple Interaction coming from the source interactive part. Subsets *involving interaction* Subsets *target connection*

6.3.2.9 Message

Package: Interactive Behavior Model
isAbstract: No
Generalization: "Simple Interaction"

Description

A **Message** is a kind of **Simple Interaction** that has an **Interaction Role** as one of its **Interactive Parts**.

Constraint

[1] At least one of the **Interactive Parts** of a **Message** must be an **Interaction Role**.

BPMN Notation

The shape of **Message** depends on its sub-types.

The line connecting a **Message** to its **Interaction Role**(s) MUST have an open arrowhead and MUST be drawn with a dashed single black line.

The line connecting a **Message** to other kinds of **Interactive Part** MUST have a solid arrowhead and MUST be drawn with a solid single line.

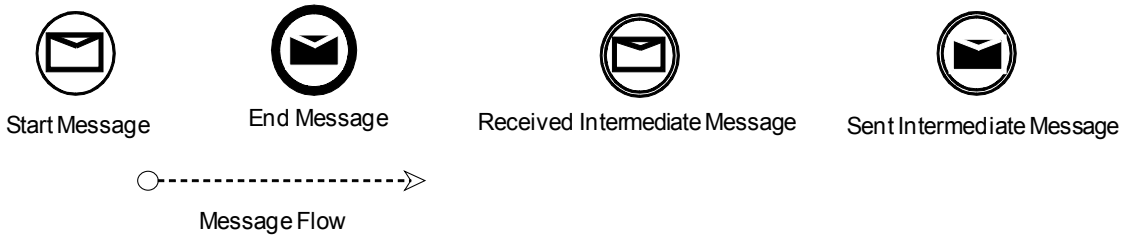


Figure 35 - Message Notation

6.3.2.10 Message Flow

Package: Interactive Behavior Model
isAbstract: No
Generalization: "Message"

Description

A **Message Flow** is a **Message** that has no succession to any other **Message** or **Event Part**. Such a **Message** doesn't have any influence on the course of its owning **Interactive Behavior**.

BPMN Notation

A **Message Flow** is a line with an open arrowhead that MUST be drawn with a dashed single black line.

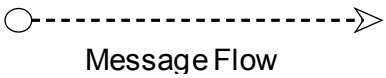


Figure 36 - Message Flow Notation

6.3.2.11 Received Intermediate Message

Package: Interactive Behavior Model
isAbstract: No
Generalization: "Message"

Description

A **Received Intermediate Message** is a **Message** that has the following additional characteristics:

- Its **source interactive part** is an **Interaction Role**.
- It has a **next succession**.

BPMN Notation

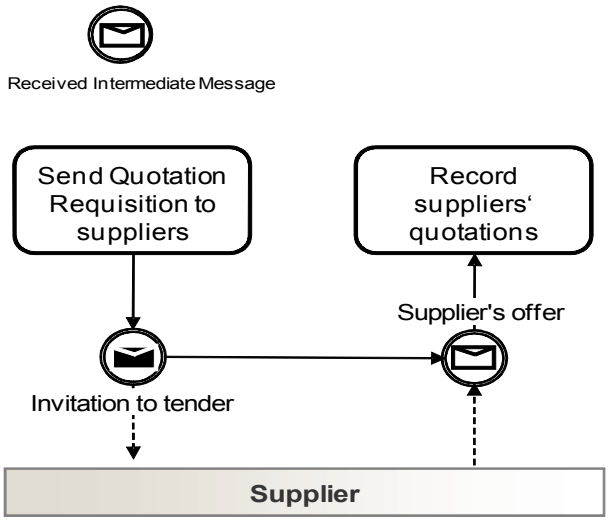


Figure 37 - Received Intermediate Message Notation

6.3.2.12 Sent Intermediate Message

Package: Interactive Behavior Model
isAbstract:
Generalization: "Message"

Description

A **Sent Intermediate Message** is a **Message** that has the following additional characteristics:

- Its **target interactive part** is an **Interaction Role**.
- It has a **previous succession**.

BPMN Notation

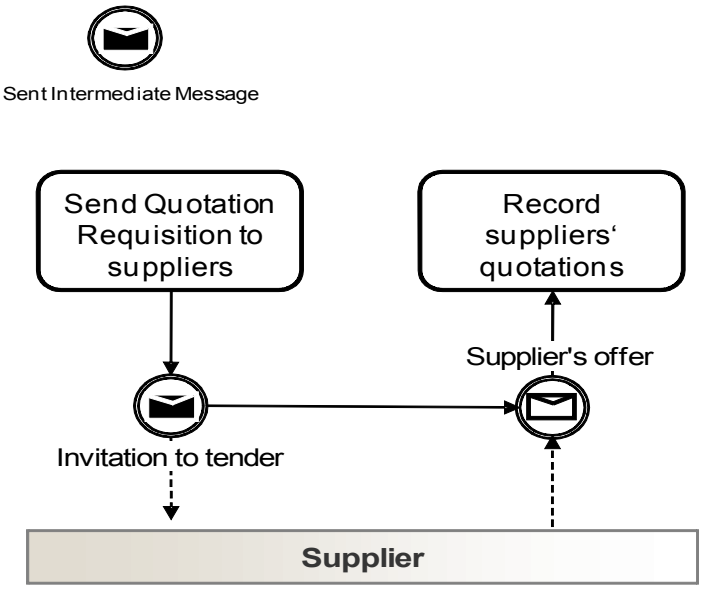


Figure 38 - Sent Intermediate Message Notation

6.3.2.13 Simple Interaction

Package: Interactive Behavior Model
isAbstract: No
Generalization: "Directed Part Connection" "Interaction"

Description

A **Simple Interaction** is a kind of **Interaction** in which something is "transferred" from individuals playing one interactive part to individuals playing another interactive part. For example, a document, phone number, or package may be transferred from one department to another in a company. The transferred items must conform to a **Type** specified by the **Simple Interaction**. A **Simple Interaction** can have an **Expression** to change the item that arrives at the target based on the item flowing from the source. For example, a transformation may retrieve the zip code from an address flowing from the source to deliver the zip code to the target.

Simple Interactions in user (M1) models are always typed by the **Behavior Occurrence** (see user library **Behavior Library**). This gives them the standard **Event Parts**, such as for start and end, so the **Simple Interactions** can be ordered within an **Interaction Protocol**. This is different from the type of thing transferred.

Simple Interactions can refer to **Simple Interactions** inside the **Interactive Parts** being connected. This means the transferred thing is passed along through chains of **Simple Interactions** from inside to outside the parts, or the other way.

Associations

source interactive part : Interactive Part [1]	Interactive Part that is the source of the Simple Interaction. Subsets *involved interactive part* Subsets *source*
target interactive part : Interactive Part [1]	Interactive Part that is the target of the Simple Interaction. Subsets *involved interactive part* Subsets *target*
transferred item type : Type [1]	Specifies the type of the item transferred by the Simple Interaction.

transformation expression : Expression [0..1]	Expression used to transform the item that arrives at the target based on the item flowing from the source. For example, a transformation may retrieve the zip code from an address flowing from the source to deliver the zip code to the target. Subsets *ownedElement*

6.3.2.14 Start Message

Package: Interactive Behavior Model
isAbstract: No
Generalization: "Message"

Description

A **Start Message** is a **Message** that has the following additional characteristics:

- Its **source interactive part** is an **Interaction Role**.
- It has a **previous succession** to the **Start Start Event** instance of **Event**.
- This **Succession** is an **Immediate Succession**.

The receipt of the **Start Message** creates a new execution of the process.

BPMN Notation

Notation for **Start Message** or **Simple Interaction** categorized as a **Start Message**.

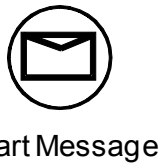


Figure 39 - Start Message Notation

6.4 Activity Model

6.4.1 Introduction

The Activity Model is for capturing orchestrations in way that facilitates modification as boundaries of process of business change, for example, due to insourcing, outsourcing, mergers, and acquisitions. It uses interactions to represent inputs and outputs, enabling choreographies to be specified between the process and its environment, as well as between the performers responsible for steps in the process. The Activity Model is the basis for the BPMN model in BPDM (see the BPMN Extension).

In the Activity Model, Processes are Interactive Behaviors that have:

- Boundaries with which processes interact to get inputs and provide outputs (Process Interaction Boundary).
- Performers for steps in the process, including a performer for the entire process (Performer Role and Processor Role).

- Steps that interact with each other and the process boundary, and invoke other processes (Activity and Embedded Process).
- Embedded processes for loops, with loop control features (Activity Loop and its subtypes).
- Holders hold flowing items (Holders).
- Steps for generating process lifecycle events, such as for errors and aborts.
- Derivations from other processes (Substitutable Derivations).

Process Interaction Boundaries and Processor Roles are the two top-level elements in Processes. The first represents entities in the environment of the process and the other the actors responsible for the process itself. They are Interactive Parts, enabling Simple interactions between them to show the inputs and outputs of a process (see the Simple Interaction Model). Inputs are simple interactions that have the boundary as source and the processor as target (or an activity in the processor, see below), and outputs have the processor as source (or an activity in the processor), and the boundary as target. The transferred item type of simple interactions specifies the kind of thing that is input or output. These interactions can be ordered in time to specify when the process is expecting its inputs and when it will provide its outputs. Multiplicities on the interactions specify how many individuals of the item type are required or allowed to be input and output by the process (see the Composition Model).

Performer Roles are Part Groups showing the responsibility of Actors for steps in the process (see Activity below). Processor Roles are actually just top-level Performer Roles, enabling them to delegate responsibility for a subset of the process steps to Performer Roles, which in turn can delegate smaller subsets to other Performer Roles. Processor Roles and Performer Roles are also Typed Parts, for specifying Actors that can play the roles. Actors are Classifiers, to specify requirements on them, such as having certain skills or budget.

Performer Roles are also Interactive Parts that can have interactions with each other as well to the boundary. This is useful when the boundaries of the process change, for example, due to outsourcing or insourcing. For outsourcing, the steps a performer role is responsible for are separated out into another process. The interactions between the performer's steps and the steps of other performers become the interactions in the protocol between the performers. This establishes a service contract for the outsourced steps in the activity. Role Realizations are Elements for showing which processes satisfy the contract. For insourcing, some of the interactions to the boundary become interactions with a performer role. This establishes the requirements on designing the steps that the performer will be responsible for.

Activities are:

- Behavior Steps, enabling them to have start and end events, be ordered in time by successions, and nest sub processes (see the Behavior Model).
- Typed Parts (due to being Behavior Steps), where the type is another Process. For Simple Activities the sub processes have no sub activities, for Sub process Activities they do.
- Interactive Parts to support simple interactions with other activities and the boundary for inputs and outputs (see the Simple Interaction Model).

Activities connected by Simple Interactions use Simple Interaction Bindings to specify which interactions in the sub processes will flow between the activities (see the Simple Interaction Model). For example, one activity might be for a process that outputs a document with an interaction to its boundary, and another activity might be for a process that inputs a document with an interaction from its boundary. These processes might output and input many other documents. The simple interaction bindings on the interaction between the activities identify which of the interactions in the sub processes are the ones that support the flow between the activities. The bindings ensure that whenever the document flows during the enactment of the process, that the exact same M0 flow plays all three interaction parts simultaneously: the output interaction in one sub process, the interaction between the activities, and the input interaction in the other sub process. In many cases, the simple interaction bindings can be derived from the types of things flowing, so the modeler does not need to specify them manually. For example, if the sub processes have only one interaction outputting and inputting a document, then simple interactions transferring documents between the sub processes will bind to those internal interactions.[6]

Embedded Processes are Behavior Step Groups that enclose Activities, enabling embedded processes to have their own lifecycle events, such as starting and ending, that interact with the enclosed activities. Every embedded process

[6] Simple interaction bindings can be derived if the interaction between the activities has a transferred item type that is the same or a super type of exactly one output interaction flow on the source end of the interaction, or has a transferred item type that is the same or a subtype of exactly one input interaction flow on the target end of the interaction.

has the Abort Group Connection applied to it (see the Behavior Model). This ensures the enclosed steps abort when the group does.

Activity Loops are Embedded Processes that can execute their enclosed activities as a group multiple times.

The process can proceed past the loop in several ways:

- After all sub executions are complete, with a succession that has the loop as the source.
- After each sub execution, with succession that has the ‘Iteration End’ event part as an internal source. This part is defined in a user (M1) library in the Activity Model, typed by the ‘Iteration End’ event also defined in the library.
- After the first sub execution to complete, with a succession that has the ‘Iteration End’ event part as an internal source, and a guard evaluating to the string "first iteration."
- After each sub execution, but depending on conditions, with a succession that has the ‘Iteration End’ event part as an internal source, and a guard specified by the modeler.

Activity Loops are of two kinds:

- Conditional Loops execute their enclosed activities multiple times as a group while a specified condition is true. If the condition is never true, the enclosed activities are never executed. The multiple sub executions are sequential.
- MultiInstance Loops execute their enclosed activities as a group a certain number of times, as specified by the modeler in an integer-valued expression evaluated at the time the loop begins executing. MultiInstance Loops support the option of sequential or parallel sub executions.

Holders are Interactive Parts for storing items temporarily as they flow through the process. For example, a document, phone number, or package can flow along simple interactions, into a holder for some period, and flow out later. The type of the holder is the type of thing it can hold.

Substitutable Derivations are Derivations of one process from another that do not alter the interactions with the boundary (see the Composition Model).

6.4.2 Metamodel Specification

The **Activity Model** is for capturing orchestrations in way that facilitates modification as boundaries of process of business change, for example, due to insourcing, outsourcing, mergers, and acquisitions. It uses interactions to represent inputs and outputs, enabling choreographies to be specified between the process and its environment, as well as between the performers responsible for steps in the process. The **Activity Model** is the basis for the BPMN model in BPDM (see the BPMN Extensions).

6.4.2.1 Activity Model Diagram

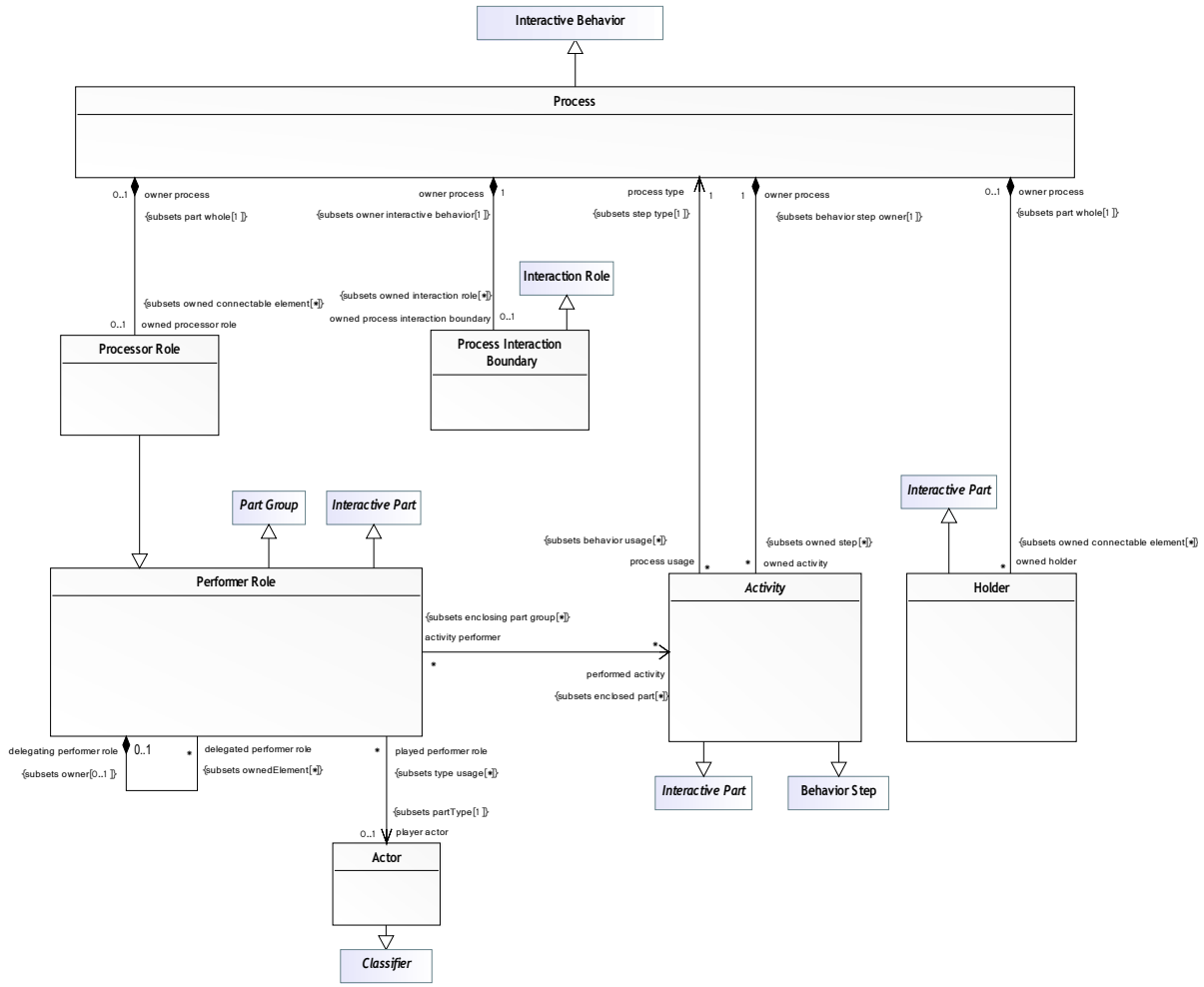


Figure 40 - Activity Model Diagram

6.4.2.2 Activity Model Library: Simple Process instances

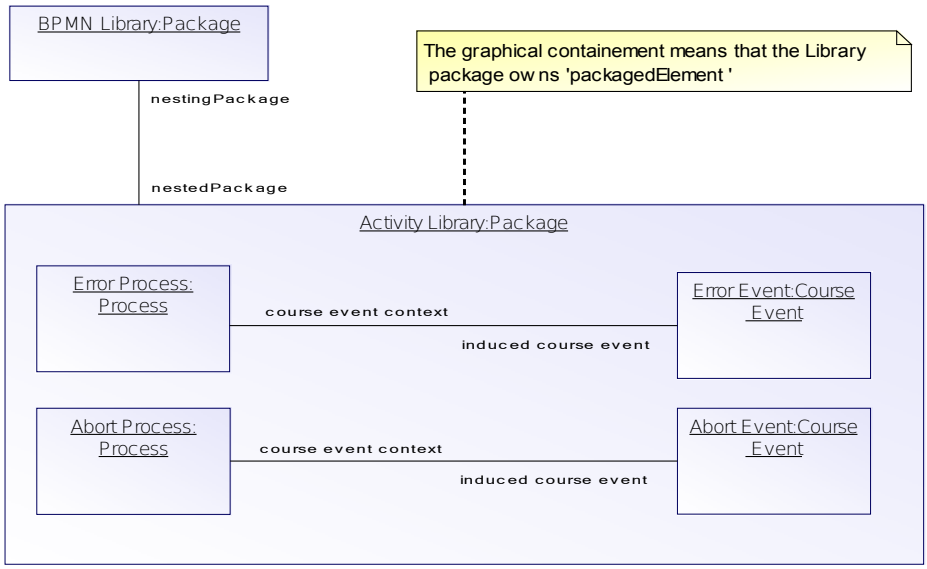


Figure 41 - Activity Model Library: Simple Process instances

6.4.2.3 Activity Categories Diagram

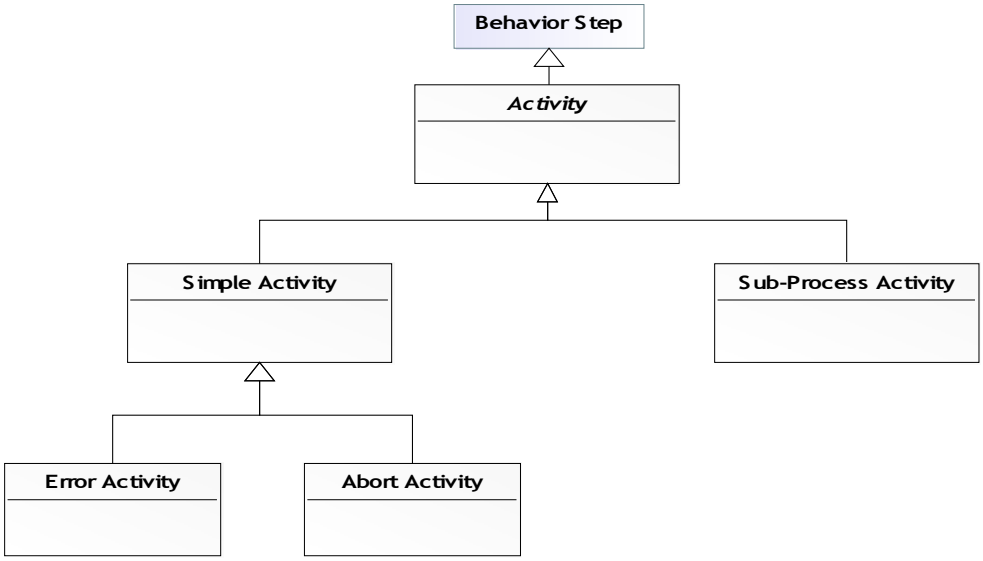


Figure 42 - Activity Categories Diagram

6.4.2.4 Activity Model Library: Loop Happening instance

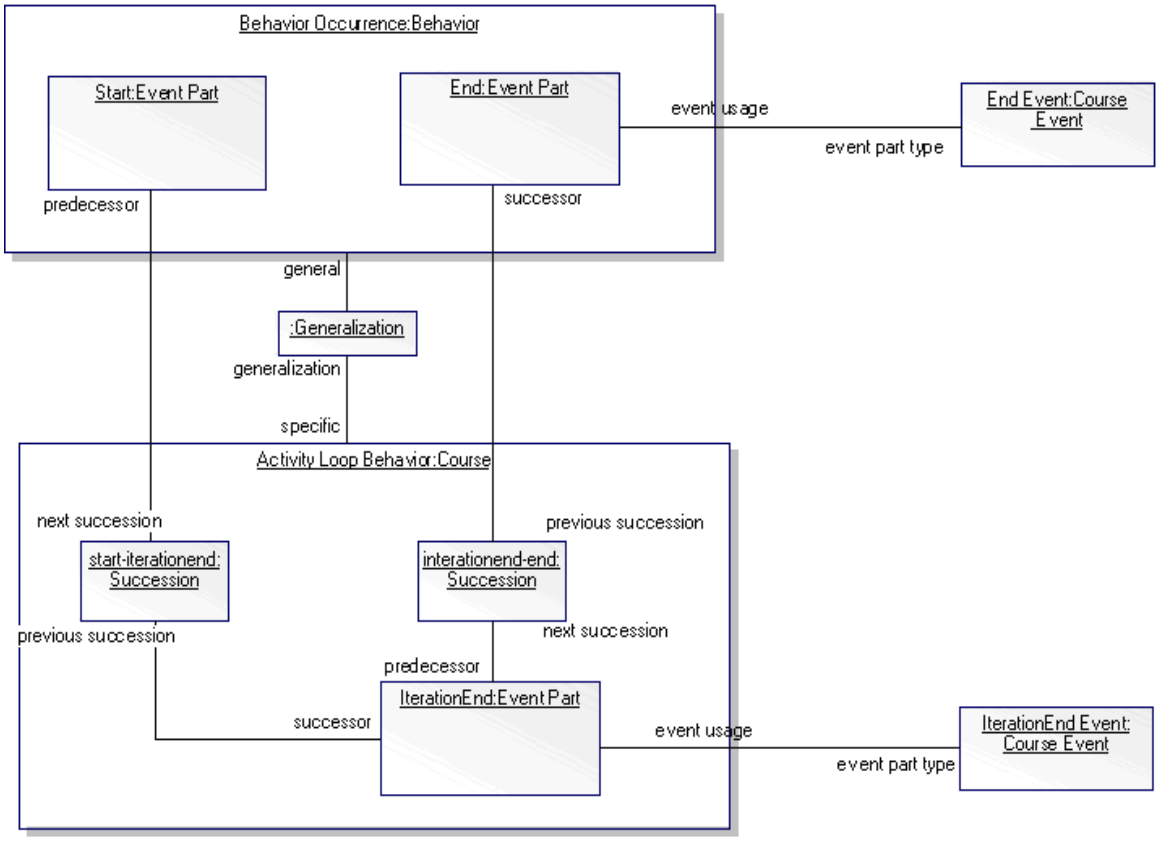


Figure 43 - Activity Model Library: Loop Happening instance

6.4.2.5 Embedded Process Diagram

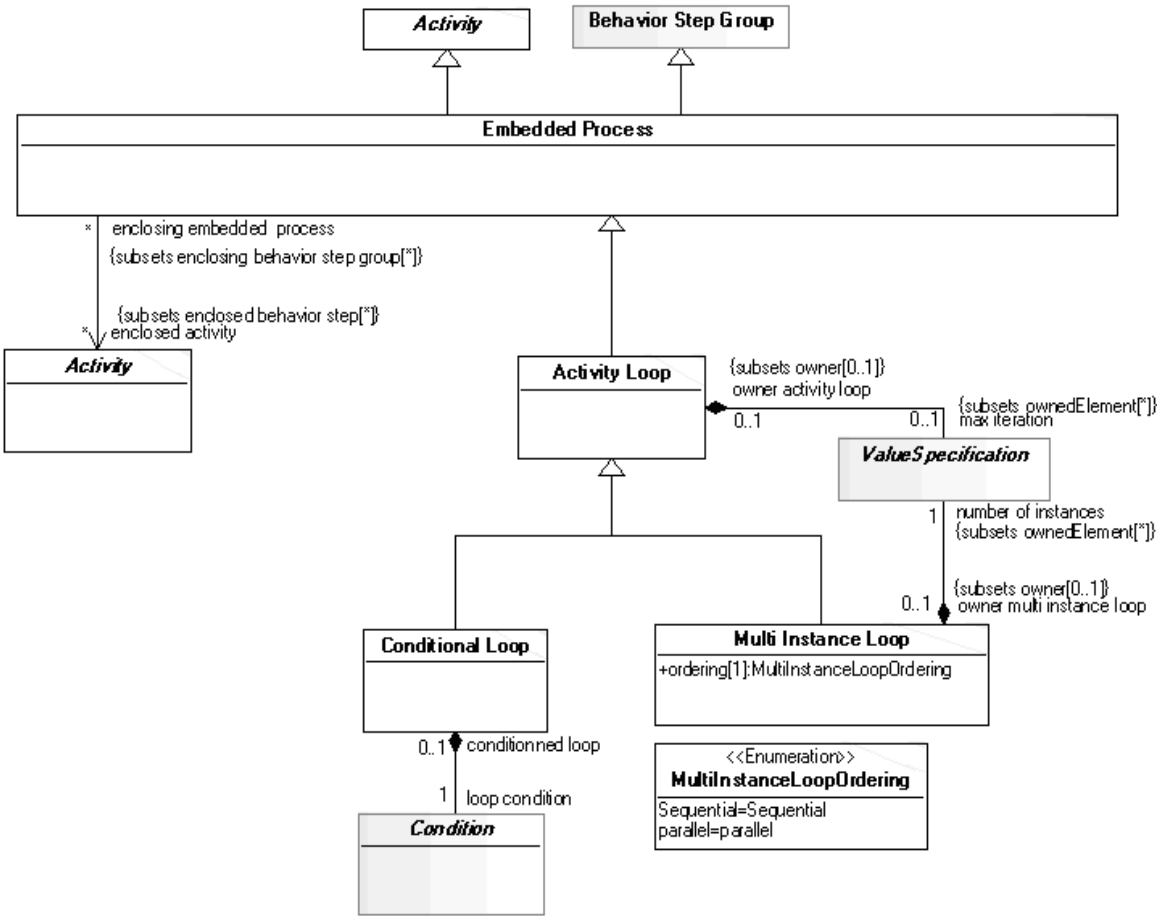


Figure 44 - Embedded Process Diagram

6.4.2.6 Process Derivation Diagram

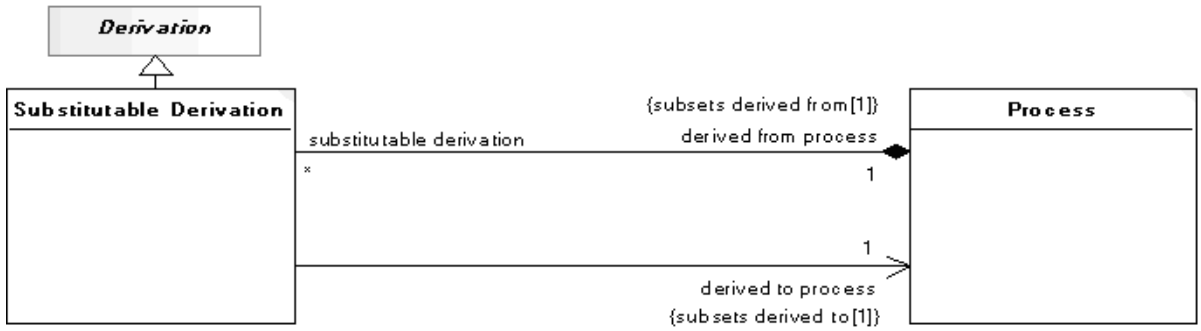


Figure 45 - Process Derivation Diagram

6.4.2.7 Role Realization Diagram

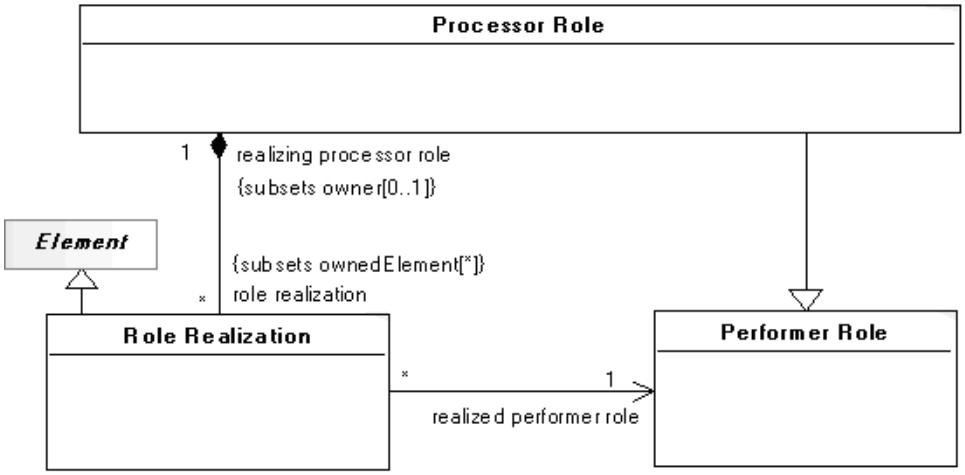


Figure 46 - Role Realization Diagram

6.4.2.8 Abort Activity

Package: Activity Model
isAbstract: No
Generalization: "Simple Activity"

Description

An **Abort Activity** is a **Simple Activity** that interrupts the course of a **Process**. All activities in the **Process** should be immediately ended. The **Process** is ended without compensation or event handling. The type of all **Abort Activity(ies)** must be **Abort Process** provided by the **BPMN Library** for the Activity Model (**Activity Library**).

BPMN Notation

This symbol can alternatively represent:

1. **Event Part** typed by the **Abort Event** instance of **Event**.
2. An **Abort Activity**

Figure 47 - Abort Activity Notation or 'Abort' Behavioral Change Part

6.4.2.9 Activity

Package: Activity Model
isAbstract: Yes
Generalization: “Behavior Step” “Interactive Part”

Description

An **Activity** is a kind of **Behavior Step** that activates a **Behavior** (it operates over time) in the context of a **Process**. It can:

- be ordered in time by **Succession**,
- operate under the responsibility of a **Performer Role**,
- activate a sub-process or be a simple task that start and stop.

An **Activity** is also an **Interactive Part** that receives its inputs and outputs through **Interactions** coming from other **Interactive Part**s in the **Process** (**Activity**, **Interaction Role**, **Performer Role**, **Holder**).

Associations

process type : Process [1]	Type of the Activity Subsets *step type*

BPMN Notation

An Activity is represented by a rounded corner rectangle that MUST be drawn with a single thin black line.

Figure 48 - Activity Notation

6.4.2.10 Activity Loop

Package: Activity Model
isAbstract: No
Generalization: “Embedded Process”

Description

An **Activity Loop** is an **Embedded Process** that can execute its enclosed activities multiple times. The process can proceed past the loop in several ways:

- After all subexecutions are complete, with a succession that has the loop as the source.
- After each subexecution, with succession that has the iterationEnd behavior part as an internal source. This part is defined in a user (M1) library in the Activity Model, typed by the IterationEnd change also defined in the library.
- After the first subexecution to complete, with a succession that has the **IterationEnd** as an internal source, and a guard evaluating to the string "first iteration."
- After each subexecution, but depending on conditions, with a succession that has the iterationEnd behavior part as an internal source, and a guard specified by the modeler.

Associations

max iteration : ValueSpecification [0..1]	the maximum number of iteration Subsets *ownedElement*

BPMN Notation

An **Activity Loop** has the shape of **Activity** with a small looping indicator will be displayed at its bottom-center.

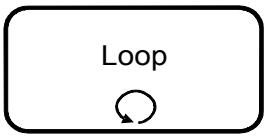


Figure 49 - Activity Loop Notation

6.4.2.11 Actor

Package: Activity Model
isAbstract: No
Generalization: "Classifier"

Description

An **Actor** is an entity that is responsible for the execution of duties specified by a **Performer Role**.
Further sub-type of **Actor** will be defined in specifications such as the Organizational Structure Metamodel (OSM) to add specific requirements such as and can as having certain skills or budget.

6.4.2.12 Conditional Loop

Package: Activity Model
isAbstract: No
Generalization: "Activity Loop"

Description

Conditional Loop is a kind of **Activity Loop** that will execute its enclosed activities multiple times as a group while a specified condition is true. If the condition is never true, the enclosed activities are never executed. The multiple subexecutions are sequential.

Associations

loop condition : Condition [1]	Condition that controls the iterations of a Conditional Loop.

6.4.2.13 Embedded Process

Package: Activity Model
isAbstract: No
Generalization: "Activity" "Behavior Step Group"

Description

An **Embedded Process** is a kind of **Behavior Step Group** that groups a set of **Activity** that, as a whole, act as a **Behavior Step**. Thereby, an **Embedded Process** is typed by a **Course** that defines its **start change** and a **finish change**. As any **Behavior Step,** an **Embedded Process** can be interrupted or constrained in its **Course** course.

Associations

enclosed activity : Activity [*]	Activity that is part of the Embedded Process. Subsets *enclosed behavior step*

Constraint

[1] An **enclosed activity** of an **Embedded Process** must belong to the **Process** owning the **Embedded Process**.

BPMN Notation

A Sub-Process Activity shares the same shape as the Activity object, which is a rounded rectangle. A Sub-Process Activity is a rounded corner rectangle that MUST be drawn with a single thin black line. If the Sub-Process Activity is also a transaction, it has a boundary drawn with a double line.

The Sub-Process Activity can be in a collapsed view that hides its details or a Sub-Process can be in an expanded view that shows the details of its Process Type.

In the collapsed form, the Sub-Process Activity uses a marker to distinguish it as a Sub-Process Activity, rather than a Simple Activity. The Sub-Process Activity marker MUST be a small square with a plus sign (+) inside. The square MUST be positioned at the bottom center of the shape.

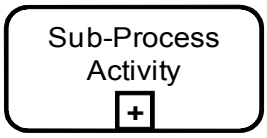


Figure 50 - Collapsed Sub-Process Activity Notation

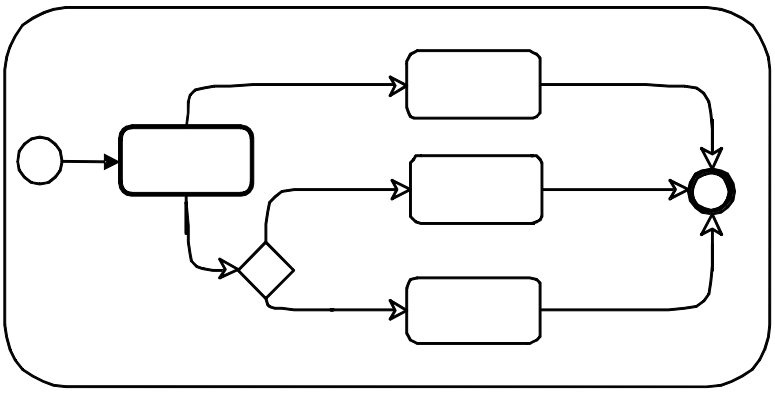


Figure 51 - Uncollapsed Sub-Process Activity Notation

6.4.2.14 Error Activity

Package: Activity Model
isAbstract: No
Generalization: "Simple Activity"

Description

An **Error Activity** is a kind of **Simple Activity** that produces an **Error Event** and that ends its enclosing **Course**. In the case where the **Error Activity** is part of an **Embedded Process**, the ended **Course** is this **Embedded Embedded Process**, otherwise the ended **Course** is the **Process** that owns the **Error Activity**.

BPMN Notation

This symbol can alternatively represent:

1. **Event Part** typed by the **Error Event** instance of **Event**.
2. An **Error Activity**

Figure 52 - Error Activity Notation or 'Error' Behavioral Event Step

6.4.2.15 Holder

Package: Activity Model
isAbstract: No
Generalization: "Interactive Part"

Description

A **Holder** is an **Interactive Part** storing items temporarily as they flow through the **Process**. For example, a document, phone number, or package can flow along simple interactions, into a holder for some period, and flow out later. The type of the **Holder** is the type of thing it can hold.

Non Normative Notation

A Holder is represented by a can that MUST be drawn with a single thin black line.

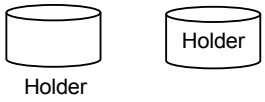


Figure 53 - Holder Notation

6.4.2.16 LoopTestTime

Package: Activity Model
isAbstract: No

Description

Enumeration of the following literal values:

after:

before:

6.4.2.17 Multi Instance Loop

Package: Activity Model
isAbstract: No
Generalization: "Activity Loop"

Description

Multi Instance Loop is a kind of **Activity Loop** that will execute its enclosed activities as a group of times, as specified by the **number of instances ValueSpecification** evaluated at the time the loop begins executing. A **Multi Instance Loop** supports the option of sequential or parallel subexecutions as specified by its **ordering** attribute.

Attributes

ordering: MultiInstanceLoopOrdering [1]

Associations

number of instances : ValueSpecification [1]	Number of instance of iteration. Subsets *ownedElement*

6.4.2.18 MultiInstanceLoopOrdering

Package: Activity Model
isAbstract: No

Description

Enumeration of the following literal values:

parallel:

Sequential:

6.4.2.19 Performer Role

Package: Activity Model
isAbstract: No
Generalization: “Interactive Part” “Part Group”

Description

A **Performer Role** is a **Part Group** that takes responsibility of performing activities in the process. Being an **Interactive Part**, a **Performer Role** also has responsibilities to fulfill **Interactions** that it is involved with other **Performer Roles** or with **Interaction Roles** at the boundary of the **Process**. A **Performer Role** is a **Typed Part** for specifying **Actor** that can play the role at process enactment.

A **Performer Role** can be decomposed into sub **Performer Role** to delegate responsibility for a subset of its activities or interactions. A **Performer Role** may have a realization as defined by a **Role Realization** that further specifies how the **Performer Role** will meet its responsibilities.

Associations

performed activity : Activity [*]	Specifies the set of Activity(ies) that are under the responsibility of the Performer Role. Subsets *enclosed part*
player actor : Actor [0..1]	Actor that, at runtime, is responsible for the execution of the responsibilities specified by the Performer Role. Subsets *partType*

BPMN Notation

A Performer Role is represented by a Lane. A lane is a sub-partition of the Pool representing the **Processor Role** of the process or a sub-partition of the Lane representing its delegating performer role.

A Lane will extend the entire length of its containing Pool or Lane, either vertically or horizontally. 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 (the Performer Role name) 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 54 - Horizontal Lane Notation

A Performer Role is represented by a Lane. A lane is a sub-partition of the Pool representing the Processor Role of the process or a sub-partition of the Lane representing its delegating performer role.

A Lane will extend the entire length of its containing Pool or Lane, either vertically or horizontally. 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 (the Performer Role name) 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.

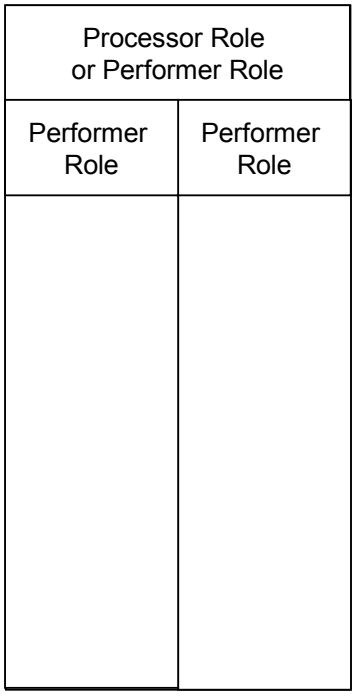


Figure 55 - Vertical Lane Notation

6.4.2.20 Process

Package: Activity Model
isAbstract: No
Generalization: "Interactive Behavior"

Description

A **Process** is a kind of **Interactive Behavior** that describes specific **Activity**(ies) to be performed, **Interaction**s to be undertaken during its execution under the authority of a **Processor Role** (or **delegated performer role**s).
The process owns the set of activities to be performed as well as the **Condition**s on when such activities will be performed and by which performer role. The process also owns the set of **Interactive Part**s that define the flow of information and other resources between activities, **Performer Role** and **Interaction Role**s.

A specific **Interaction Role** defines the set of **Interaction**s the process is responsible of: it is the **Process Interaction Boundary**. The set of **Interaction**s attached to the **Process Interaction Boundary** defines the inputs and outputs of the process.

A Process may utilize sub-processes with a **Sub-Process Activity** as well as be used in the context of other processes in the same way.

Associations

owned activity : Activity [*]	Activity owned by the Process. Subsets *owned step*

owned holder : Holder [*]	Holder owned by the Process. Subsets *owned connectable element*
owned process interaction boundary : Process Interaction Boundary [0..1]	Specifies the set of Interactions the process is responsible for. This set of Interactions defines the inputs and outputs of the process. Subsets *owned interaction role*
owned processor role : Processor Role [0..1]	Processor Role of the Process. Subsets *owned connectable element*
substitutable derivation : Substitutable Derivation [*]	

Non Normative Notation

Each process diagram has a contents area. As an option, it may have a frame and a heading as shown in the following figure. The frame is a rectangle. The frame may be omitted and implied by the border of the diagram area provided by a tool. In case the frame is omitted, the heading is also omitted.

The diagram contents area contains the graphical symbols. The heading is a string contained in name tag (rectangle with cutoff corner) in the upper leftmost corner of the rectangle, with the following syntax: <process name>.

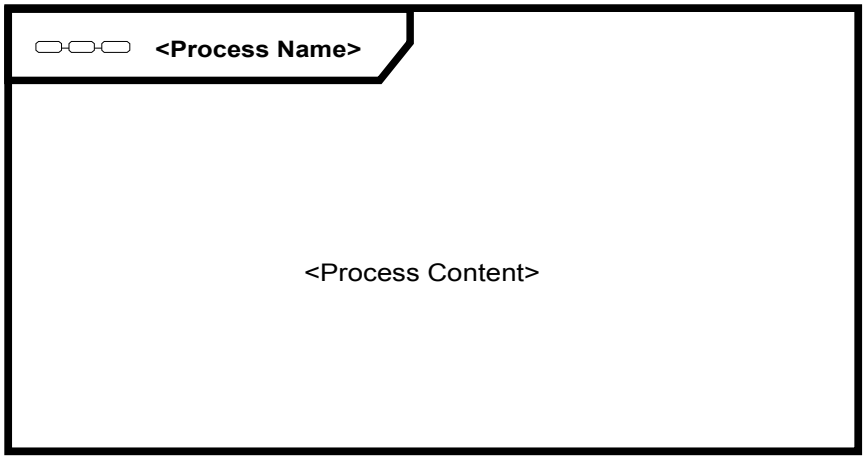


Figure 56 - Process Diagram

6.4.2.21 Process Interaction Boundary

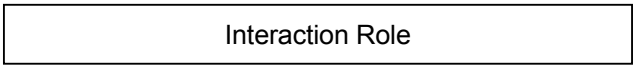
Package: Activity Model
isAbstract: No
Generalization: “Interaction Role”

Description

The **Process Interaction Boundary** is the **Interaction Role** through which a **Process** interacts to get its inputs and deliver its outputs. The process is responsible to fulfill all **Interactions** attached to the **Process Interaction Boundary**.

BPMN Notation

A "black box pool" is a pool that does not have any process details.



Interaction Role as a black box pool

Figure 57 - Interaction Role Notation

6.4.2.22 Processor Role

Package: Activity Model
isAbstract: No
Generalization: "Performer Role"

Description

A **Processor Role** is the top-level **Performer Role** responsible for all activities and interactions at the boundary of the **Process**. As all **Performer Role**s, it can delegate responsibility for a subset of the process activities and interactions to **Performer Role**s, which in turn can delegate smaller subsets to other **Performer Role**s (**delegated performer role**).

A **Processor Role** may be active or passive. An active processor will control and/or monitor the process and may manage process resources. A passive processor delegates all responsibilities to delegee role. The actor of a passive processor may be a "community," consensus body or group of actors who have agreed to work together in a particular way. The actor of an active processor must be an individual, system, or organization capable of taking action, initiating and responding to **Interaction**s, and managing resources.

Associations

role realization : Role Realization [*]	Specification of the set of Performer Role that the Processor Role is the realization of. Subsets *ownedElement*

BPMN Notation

A Processor Role is represented by a Pool. A Pool is a square-cornered rectangle that MUST be drawn with a solid single black line.

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.

The Processor Role Pool MAY be presented without a boundary.

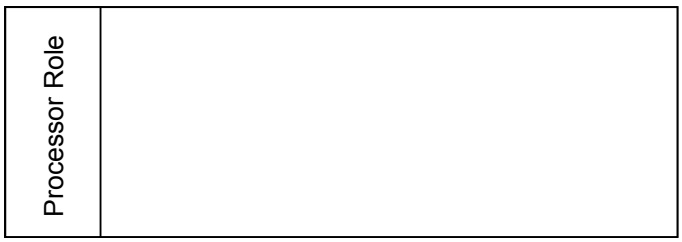


Figure 58 - Processor Role Notation

6.4.2.23 Role Realization

Package: Activity Model
isAbstract: No
Generalization: "Element"

Description

A role realization takes a realized performer role and defines a processor role and the associated process that specifies the specific process to be enacted by the specified processor role as required to meet the responsibilities of the realized performer role. A performer role may be realized by any number of processor roles as long as they each satisfy the responsibilities of the role.

Associations

realized performer role : Performer Role [1]	Performer Role that is the specification of the Role Realization.

6.4.2.24 Simple Activity

Package: Activity Model
isAbstract: No
Generalization: "Activity"

Description

A **Simple Activity** is an **Activity** which **process type** is no further composed of other activities.

Constraint

[1] A **Simple Activity** is typed by a process that has no owned activity.

BPMN Notation

An Activity is represented by a rounded corner rectangle that MUST be drawn with a single thin black line.

Figure 59 - Activity Notation

6.4.2.25 Sub-Process Activity

Package: Activity Model
isAbstract: No
Generalization: "Activity"

Description

A **Sub-Process Activity** is an **Activity** which **process type** is further composed of other activities.

Constraint

[1] A **Sub-Process Activity** is typed by a process that has owned activity.

BPMN Notation

A Sub-Process Activity shares the same shape as the Activity object, which is a rounded rectangle. A Sub-Process Activity is a rounded corner rectangle that MUST be drawn with a single thin black line. If the Sub-Process Activity is also a transaction, it has a boundary drawn with a double line.

The Sub-Process Activity can be in a collapsed view that hides its details or a Sub-Process can be in an expanded view that shows the details of its Process Type.

In the collapsed form, the Sub-Process Activity uses a marker to distinguish it as a Sub-Process Activity, rather than a Simple Activity. The Sub-Process Activity marker MUST be a small square with a plus sign (+) inside. The square MUST be positioned at the bottom center of the shape.

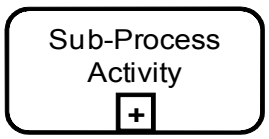


Figure 60 - Collapsed Sub-Process Activity Notation

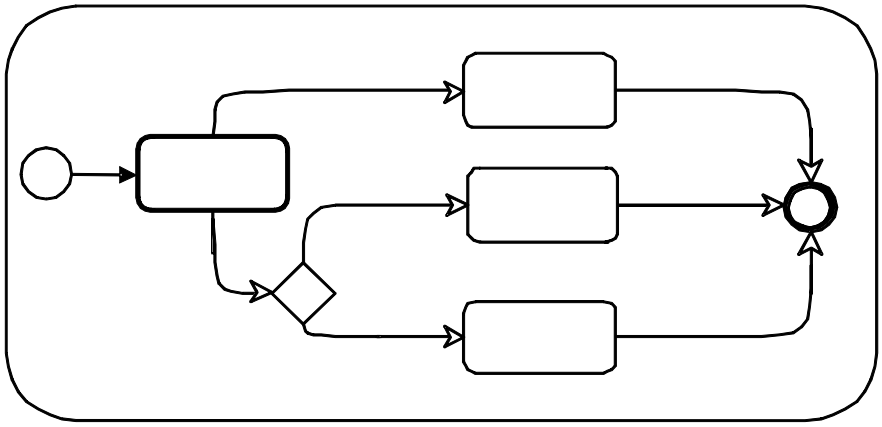


Figure 61 - Uncollapsed Sub-Process Activity Notation

6.4.2.26 Substitutable Derivation

Package: Activity Model
isAbstract: No
Generalization: “Derivation”

Description

A **Substitutable Derivation** is a kind of **Derivation** that derives one **Process** from another and that does not alter the **Interaction** at the **owned process interaction boundary**.

Associations

derived to process : Process [1]	Subsets *derived to*

6.4.2.27 Instance: Abort Process

Class: Process

Description

Links

Played End	*Opposite End*
Abort Process:course event context	*induced course event* Abort Event
Abort Process:packagedElement	*owningPackage* Activity Library

6.4.2.28 Instance: Activity Library

Class: Package

Description

Links

Played End	*Opposite End*
Activity Library:nestedPackage	*nestingPackage* BPMN Library
Activity Library:nestingPackage	*nestedPackage* Compensation Library
Activity Library:owningPackage	*packagedElement* Activity Loop Behavior
Activity Library:owningPackage	*packagedElement* IterationEnd Event
Activity Library:owningPackage	*packagedElement* Error Process
Activity Library:owningPackage	*packagedElement* Abort Process

6.4.2.29 Instance: Activity Loop Behavior

Class: Course

Description

Links

Played End	*Opposite End*
Activity Loop Behavior:step type	
Activity Loop Behavior:event part owner	*owned event part* IterationEnd
Activity Loop Behavior:owner course	*owned succession* start-iterationend
Activity Loop Behavior:owner course	*owned succession* interationend-end
Activity Loop Behavior:packagedElement	*owningPackage* Activity Library
Activity Loop Behavior:specific	*generalization* Generalization

6.4.2.30 Instance: Error Process

Class: Process

Description

Links

Played End	*Opposite End*
Error Process:course event context	*induced course event* Error Event
Error Process:packagedElement	*owningPackage* Activity Library

6.4.2.31 Instance: Generalization

Class: Generalization

Description

Links

Played End	*Opposite End*
Generalization:	*general* Behavior Occurrence
Generalization:generalization	*specific* Activity Loop Behavior

6.4.2.32 Instance: interationend-end

Class: Succession

Description

Links

Played End	*Opposite End*
interationend-end:next succession	*predecessor* IterationEnd
interationend-end:owned succession	*owner course* Activity Loop Behavior
interationend-end:previous succession	*successor* End

6.4.2.33 Instance: IterationEnd Event

Class: Course Event

Description

Links

Played End	*Opposite End*
IterationEnd Event:event part type	*event usage* IterationEnd
IterationEnd Event:packagedElement	*owningPackage* Activity Library

BPMN Notation

Marker of the **IterationEnd Event** instance of **Event**.

Iteration End Behavioral Event Instance

Figure 62 - Behavioral Event 'Iteration End'

6.4.2.34 Instance: IterationEnd

Class: Event Part

Description

Links

Played End	*Opposite End*
IterationEnd:event usage	*event part type* IterationEnd Event
IterationEnd:owned event part	*event part owner* Activity Loop Behavior
IterationEnd:predecessor	*next succession* interationend-end
IterationEnd:successor	*previous succession* start-iterationend

6.4.2.35 Instance: start-iterationend

Class: Succession

Description

Links

Played End	*Opposite End*
start-iterationend:next succession	*predecessor* Start
start-iterationend:owned succession	*owner course* Activity Loop Behavior
start-iterationend:previous succession	*successor* IterationEnd

6.5 BPMN Extensions

6.5.1 Introduction

The BPMN Extension provides additions to the Activity Model for BPMN. These provide BPMN names for special usages of BPDM concepts and additional functionality specific to BPMN. The BPMN Extension includes:

- Activities for scripts, tasks, termination, compensation, and cancelling, along with Embedded processes for transactions.
- Directives for Processes and Embedded Processes, such as adhoc directives.
- Course Control Parts specific to BPMN, such as Inclusive Merge, and specializations of BPDM course control parts, such as Inclusive Decisions.
- User (M1) library for compensation and canceling.

The descriptions of these and other elements in the BPMN Extension are available in the BPMN specification.

6.5.2 Metamodel Specification

The BPMN Extension provides additions to the Activity Model for BPMN. These provide BPMN names for special usages of BPDM concepts and additional functionality specific to BPMN.

6.5.2.1 Adhoc Extension Diagram

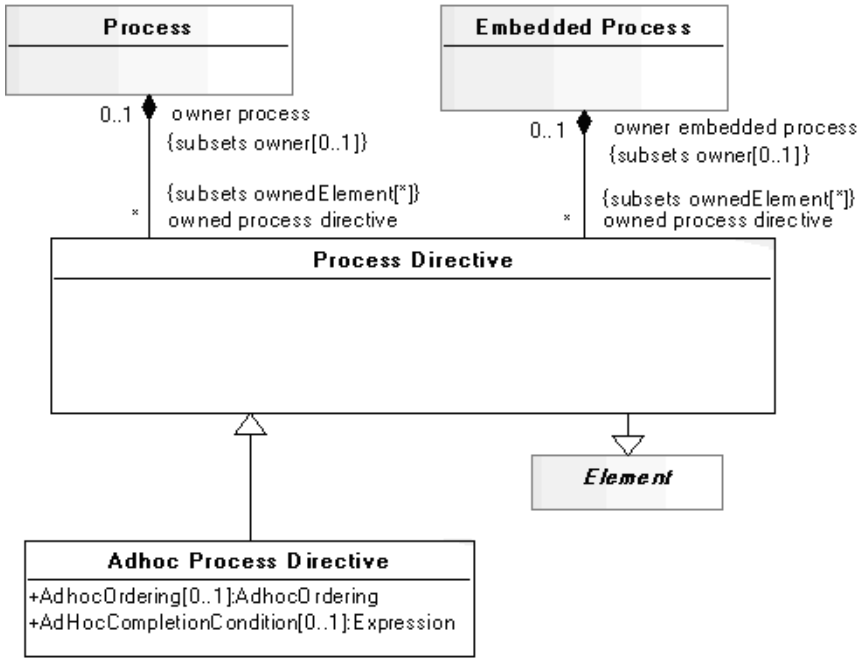


Figure 63 - Adhoc Extension Diagram

6.5.2.2 Activity Extensions Diagram

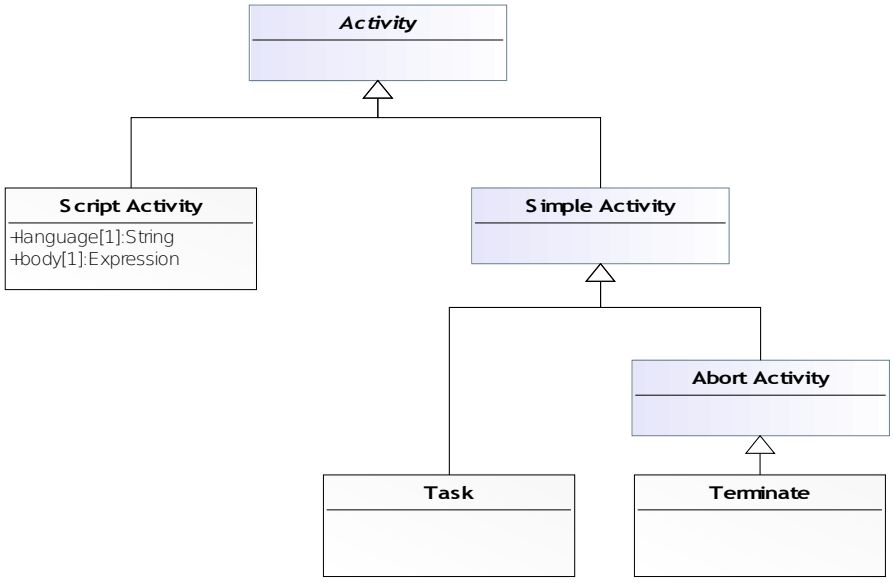


Figure 64 - Activity Extensions Diagram

6.5.2.3 Gateway Extension Diagram

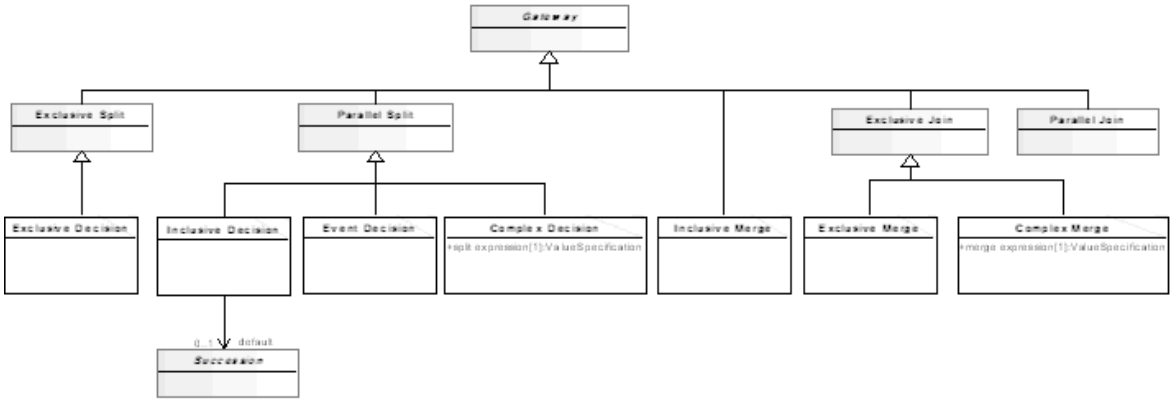


Figure 65 - Gateway Extension Diagram

6.5.2.4 BPMN Extensions Library: Compensate Process Instance

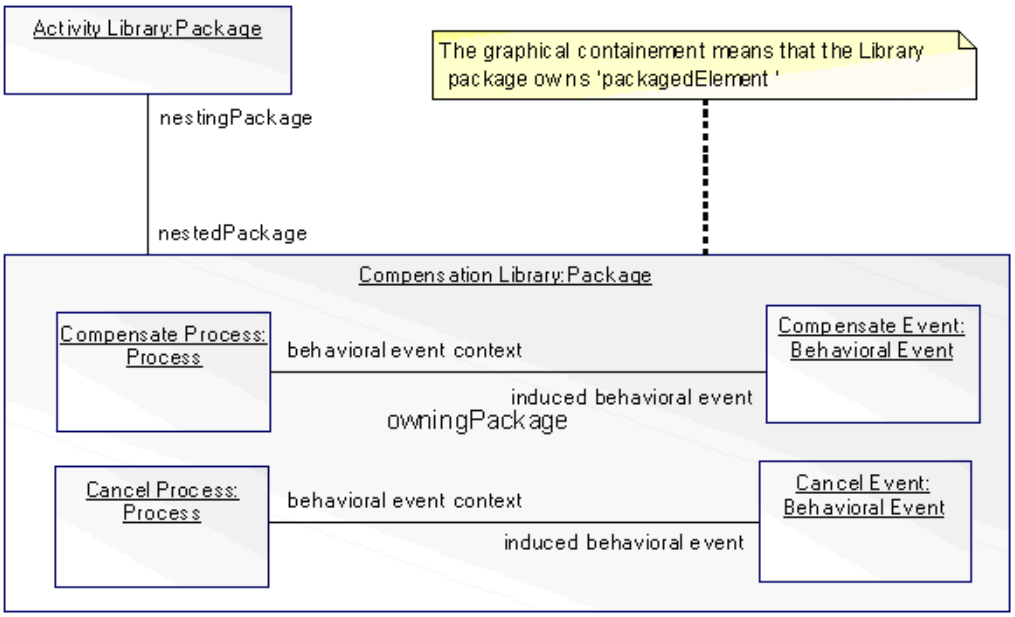


Figure 66 - BPMN Extensions Library: Compensate Process Instance

6.5.2.5 BPMN Extensions Library: BPMN Process Occurrence Instance

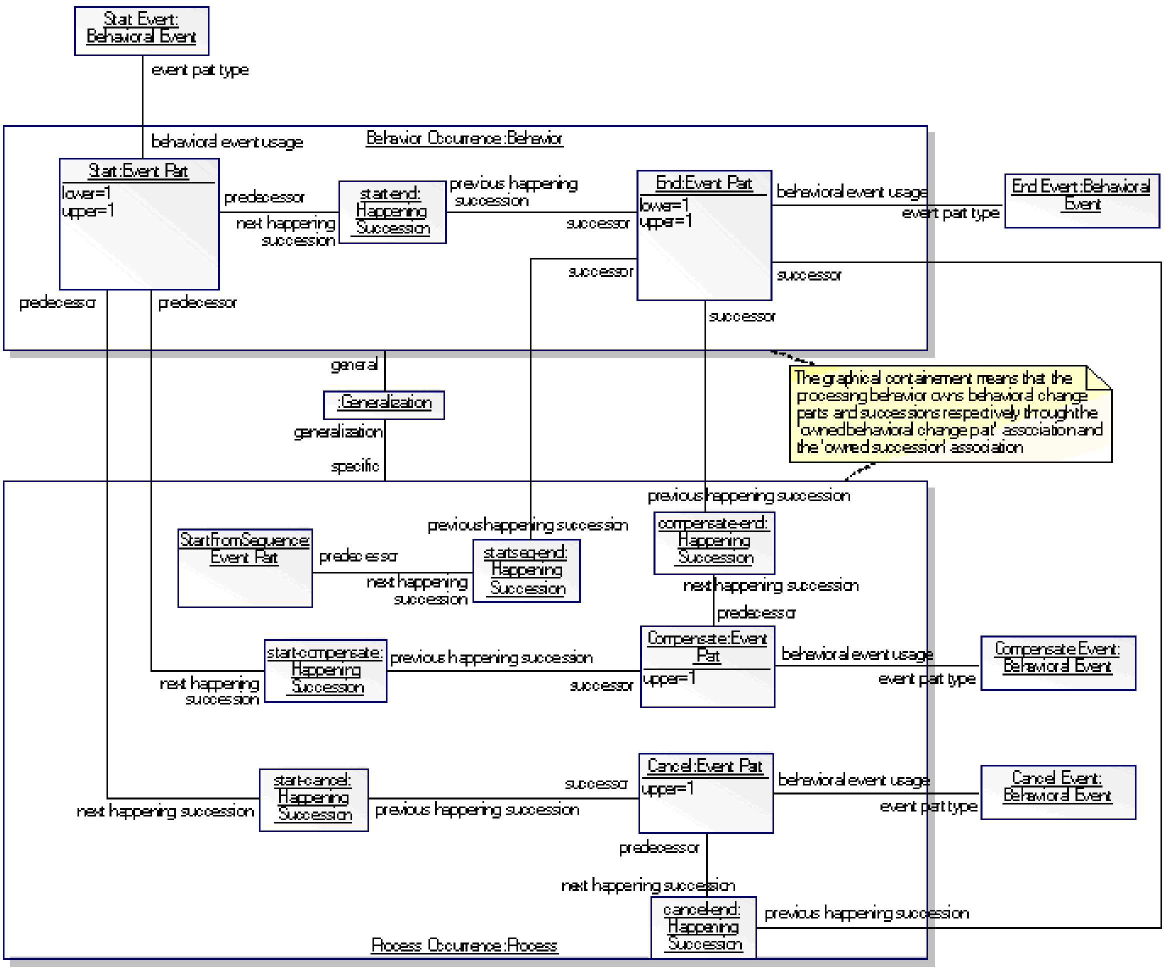


Figure 67 - BPMN Extensions Library: BPMN Process Occurrence Instance

6.5.2.6 Sequence Flow Extension Diagram

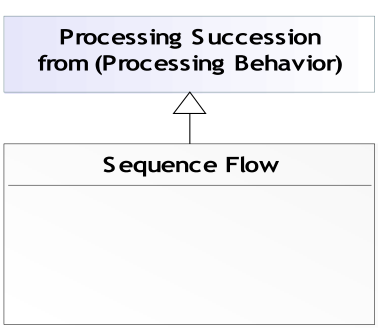
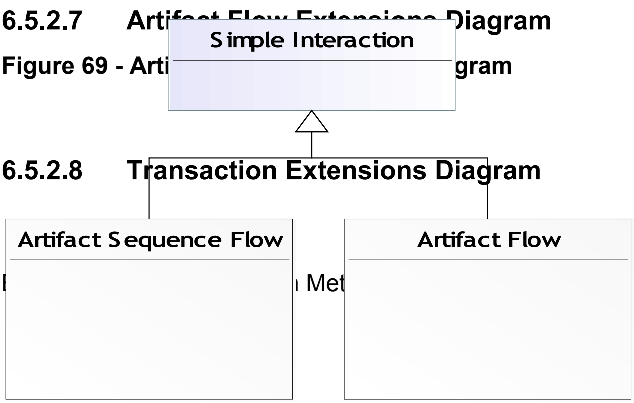


Figure 68 - Sequence Flow Extension Diagram

6.5.2.7 Arti... Diagram

Figure 69 - Arti... gram



6.5.2.8 Transaction Extensions Diagram

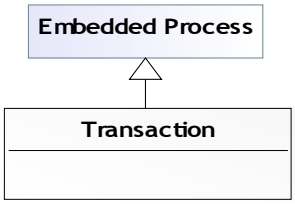


Figure 70 - Transaction Extensions Diagram

6.5.2.9 Compensation Extensions Diagram

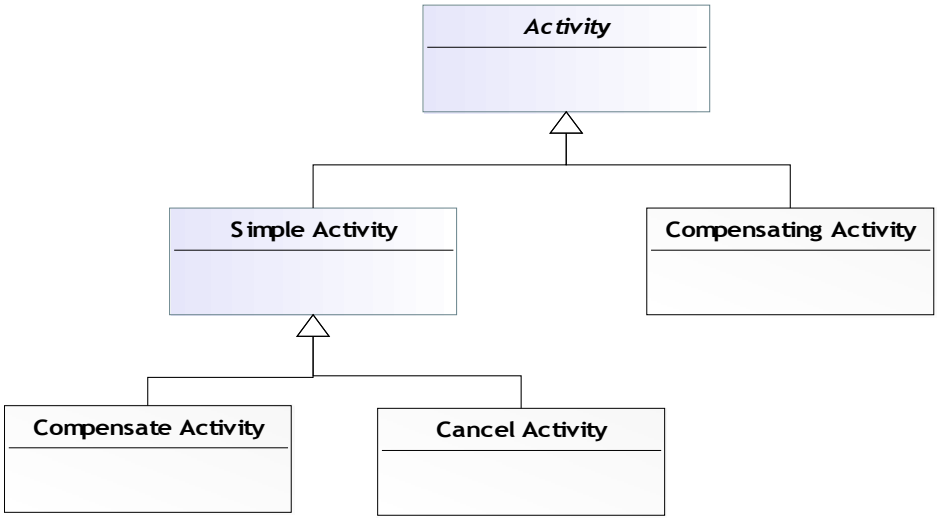


Figure 71 - Compensation Extensions Diagram

6.5.2.10 Adhoc Process Directive

Package: BPMN Extensions
isAbstract: No
Generalization: "Process Directive"

Description

Attributes

AdhocOrdering: AdhocOrdering [0..1]
AdHocCompletionCondition: Expression [0..1]

6.5.2.11 AdhocOrdering

Package: BPMN Extensions
isAbstract: No

Description

Enumeration of the following literal values:

parallel:

sequential:

6.5.2.12 Artifact Flow

Package: BPMN Extensions
isAbstract: No
Generalization: "Simple Interaction"

Description

An **Artifact Flow** is a **Simple Interaction** that has the following characteristics:

- It has a **Holder** as one of its **Interactive Parts**.
- The other interactive part is an **Activity**.
- If this **Activity** is the **source interactive part**, it has a **next processing succession** to the **Artifact Flow**.
- If this **Activity** is the **target interactive part**, it has a **previous processing succession** from the **Artifact Flow**.

6.5.2.13 Artifact Sequence Flow

Package: BPMN Extensions
isAbstract: No
Generalization: "Simple Interaction"

Description

An **Artifact Sequence Flow** is a **Simple Interaction** that has the following characteristics:

- Its **Interactive Part** are activities.
- The **source interactive part** has a **next processing succession** to the **Artifact Sequence Flow**.

BPMN Notation

An **Artifact Sequence Flow** is represented by a line with a solid arrowhead that MUST be drawn with a solid single line.

The type of the element transferred by the information flow is represented by a portrait-oriented rectangle that has its upper-right corner folded over that MUST be drawn with a solid single black line.

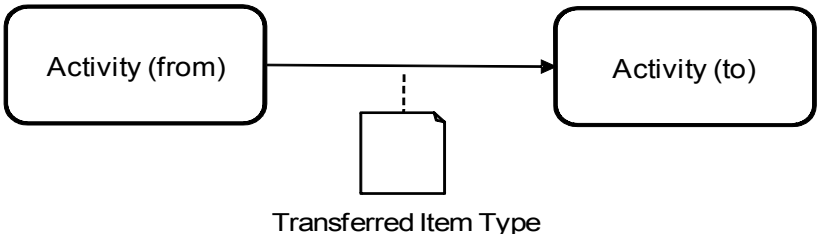


Figure 72 - Artifact Sequence Flow Notation

Non Normative Notation

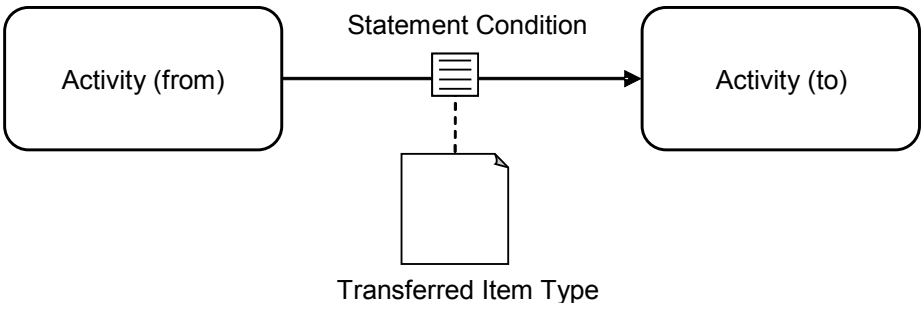


Figure 73 - Interaction Flow between Activities and Statement Condition

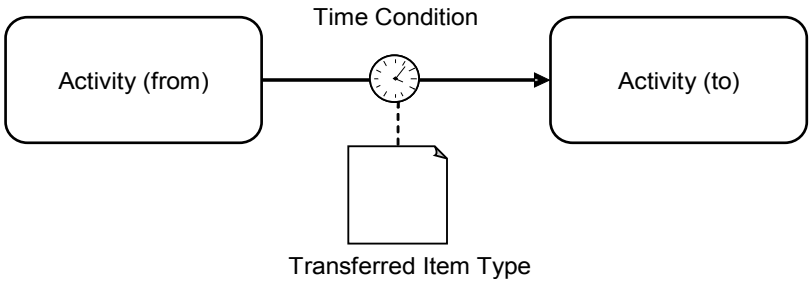


Figure 74 - Interaction Flow between Activities and Time Event Condition

6.5.2.14 Cancel Activity

Package: BPMN Extensions
isAbstract: No
Generalization: "Simple Activity"

Description

A **Cancel Activity** is a kind of **Simple Activity** that causes the **Cancel Event** of its enclosing **Behavior**.
In cases where the **Cancel Activity** is part of an **Embedded Process**, the canceled **Behavior** is this **Embedded Process**, otherwise the canceled **Behavior** is the **Process** that owns the **Cancel Activity**.

BPMN Notation

This symbol can alternatively represent:

1. **Event Part** typed by the **Cancel Event** instance of **Behavioral Event**.
2. A **Cancel Activity**

Figure 75 - Cancel Activity Notation or 'Cancel' Behavioral Event Step

6.5.2.15 Compensate Activity

Package: BPMN Extensions
isAbstract: No
Generalization: “Simple Activity”

Description

Compensate Activity is a kind of **Simple Activity** that ends a Process and indicates that a Compensation is necessary.

BPMN Notation

Figure 76 - Compensate Activity Notation

6.5.2.16 Compensating Activity

Package: BPMN Extensions
isAbstract: No
Generalization: “Activity”

Description

A **Compensating Activity** is an **Activity** that follows an **Event Monitor** conditioned by the **Compensate Event Behavioral Event**. A **Compensating Activity** cannot have successors.

Constraint

[1] A **compensating activity** cannot have **next processing succession**.

BPMN Notation

A **Compensating Activity** shares the standard activity shape with the **Compensate Event** marker displayed in the bottom center of the activity.

Figure 77 - Compensating Activity Notation

6.5.2.17 Complex Decision

Package: BPMN Extensions
isAbstract: No
Generalization: “Parallel Split”

Description

A **Complex Decision** is a **Parallel Split** that has an expression determining which outgoing **Successions** apply.

Attributes

split expression: ValueSpecification [1]	Has to evaluate to a boolean value that when evaluated to true enables the split.

BPMN Notation

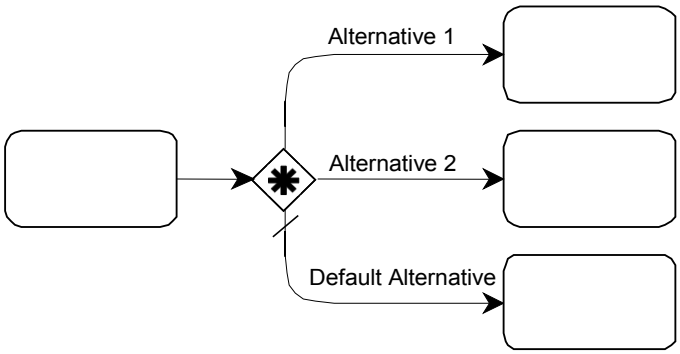


Figure 78 - Complex Decision Notation

6.5.2.18 Complex Merge

Package: BPMN Extensions
isAbstract: No
Generalization: "Exclusive Join"

Description

A **Complex Merge** is an **Exclusive Join** that has an expression determining which incoming **Successions** must apply for the merge to apply.

Attributes

merge expression: ValueSpecification [1]

BPMN Notation

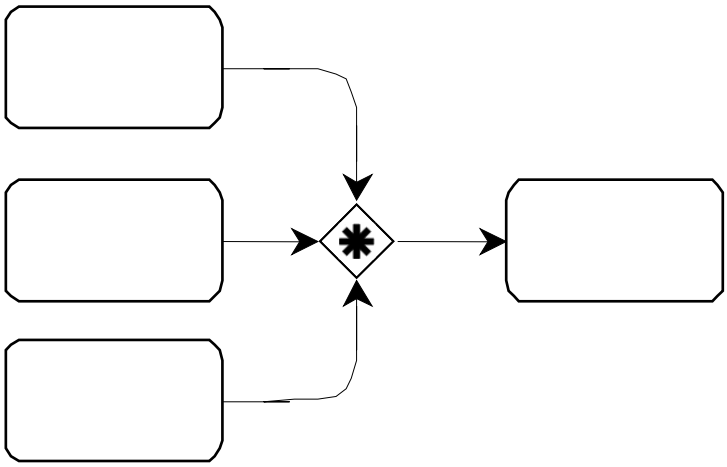


Figure 79 - Complex Join Notation

6.5.2.19 Event Decision

Package: BPMN Extensions
isAbstract: No
Generalization: "Parallel Split"

Description

An Event Decision applies a race connector to the parts on the target end of processing successions that have the event decision as source (see Processing Behavior). The targeted parts are change condition steps. To wait for incoming messages, these can include behavioral change condition steps detecting the finish of simple interactions from the boundary to the processor role.

BPMN Notation

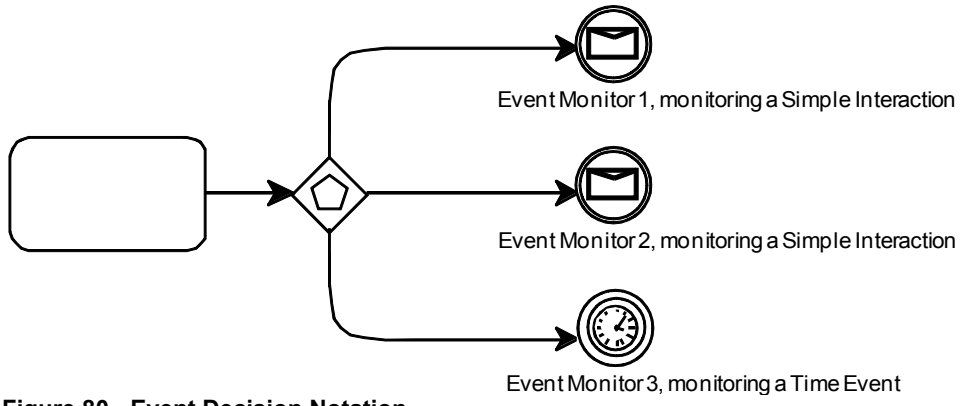


Figure 80 - Event Decision Notation

6.5.2.20 Exclusive Decision

Package: BPMN Extensions
isAbstract: No
Generalization: "Exclusive Split"

Description

Same as **Exclusive Split** but with a different name in BPMN.

BPMN Notation

The Exclusive Split shares the same basic shape, called a Gateway, of the generic **Gateway**. The Exclusive Split MAY use a marker that is shaped like an "X" and is placed within the Gateway diamond to distinguish it from other **Gateways**. This marker is not required. A Diagram SHOULD be consistent in the use of the "X" internal indicator. That is, a Diagram SHOULD NOT have some Exclusive Splits with an indicator and some Exclusive Splits without an indicator.

The **default** succession is represented by a default Marker that MUST be a backslash near the beginning of the line representing the **Succession**.

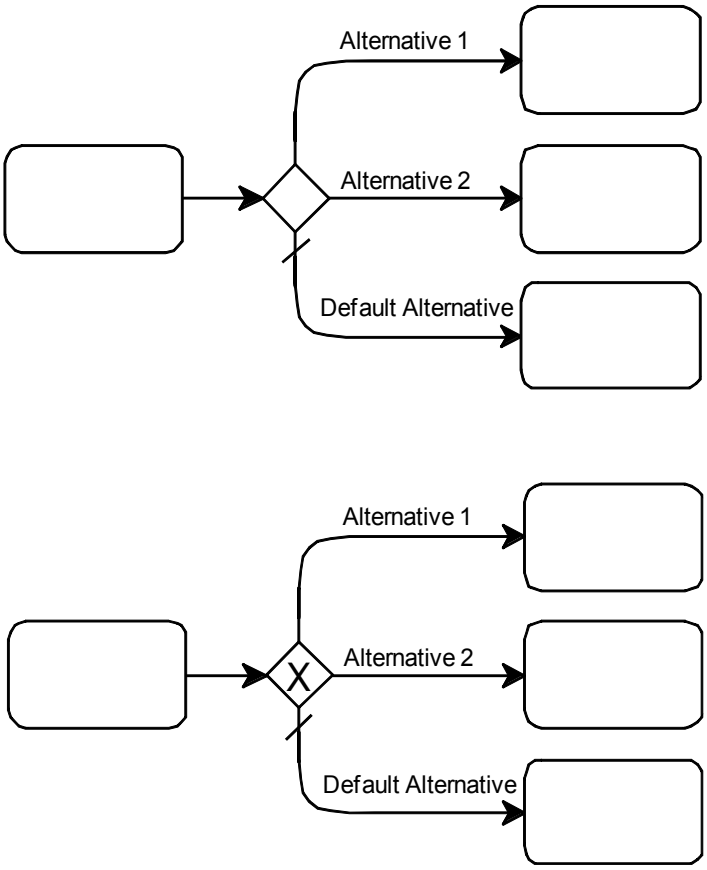


Figure 81 - Exclusive Split Notation

6.5.2.21 Exclusive Merge

Package: BPMN Extensions
isAbstract: No
Generalization: "Exclusive Join"

Description

Same as **Exclusive Join** but with a different name in BPMN.

BPMN Notation

The Exclusive Join shares the same basic shape of the generic **Gateway**.

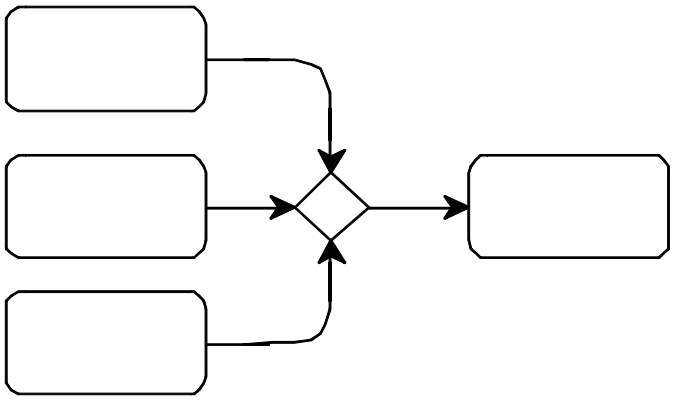


Figure 82 - Exclusive Merge Notation

6.5.2.22 Inclusive Decision

Package: BPMN Extensions
isAbstract: No
Generalization: "Parallel Split"

Description

Inclusive Decision is a **Parallel Split** that has an outgoing **Succession** specified as the default if none of the other outgoing successions apply due to their conditions.

Associations

default : Succession [0..1]	Succession enabled by default if no other next succession connected to the Inclusive Decision has been enabled.

BPMN Notation

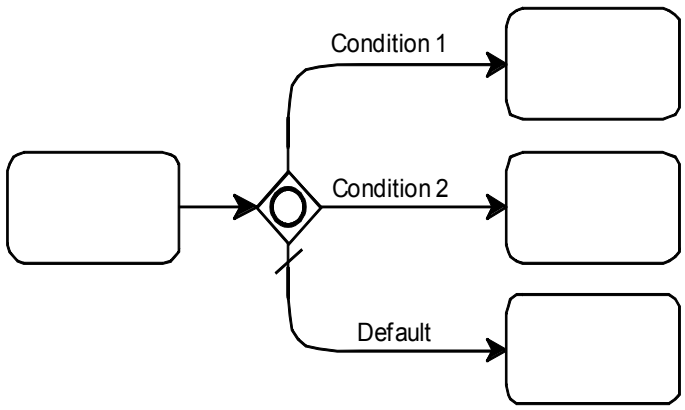


Figure 83 - Inclusive Split Notation

6.5.2.23 Inclusive Merge

Package: BPMN Extensions
isAbstract: No
Generalization: “Gateway”

Description

An **Inclusive Merge** is a **Gateway** that requires none of the upstream activities to be executing for the join to apply.

BPMN Notation

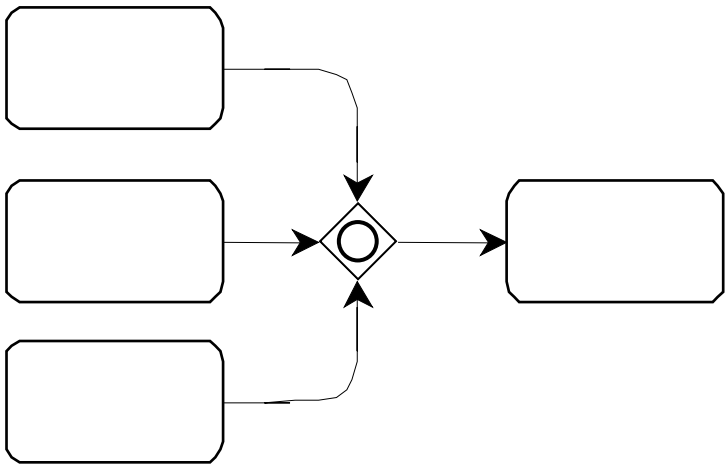


Figure 84 - Inclusive Merge Notation

6.5.2.24 Process Directive

Package: BPMN Extensions
isAbstract: No
Generalization: “Element”

Description

6.5.2.25 Script Activity

Package: BPMN Extensions
isAbstract: No
Generalization: “Activity”

Description

Attributes

language: String [1]
body: Expression [1]

6.5.2.26 Sequence Flow

Package: BPMN Extensions
isAbstract: No
Generalization: “Processing Succession”

Description

Sequences Flow is Processing Succession from one part to another (see Processing Behavior). If the source part of the succession is typed (not a control part), then if the source part has no intermediate events attached, the source

end refers to the end part (which can be omitted as the default), otherwise to the finish part. If the target part is typed, then the target part refers to the start part (which can be omitted as the default).

BPMN Notation

A Succession is a line with a solid arrowhead that MUST be drawn with a solid single line.

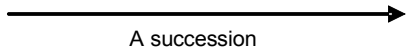


Figure 85 - Succession Notation

6.5.2.27 Task

Package: BPMN Extensions
isAbstract: No
Generalization: "Simple Activity"

Description

BPMN name for **Simple Activity**.

BPMN Notation

An Activity is represented by a rounded corner rectangle that MUST be drawn with a single thin black line.

Figure 86 - Activity Notation

6.5.2.28 Terminate

Package: BPMN Extensions
isAbstract: No
Generalization: "Abort Activity"

Description

Terminate is the BPMN name for **Abort Activity**.

BPMN Notation

This symbol can alternatively represent:

1. **Event Part** typed by the **Abort Event** instance of **Behavioral Event**.
2. An **Abort Activity**

Figure 87 - Abort Activity Notation or 'Abort' Behavioral Change Part

6.5.2.29 Transaction

Package: BPMN Extensions
isAbstract: No
Generalization: “Embedded Process”

Description

A **Transaction** is a kind of **Embedded Process** which **enclosed activity** (ies) can be rolled back by the means of an **Actor**.

BPMN Notation

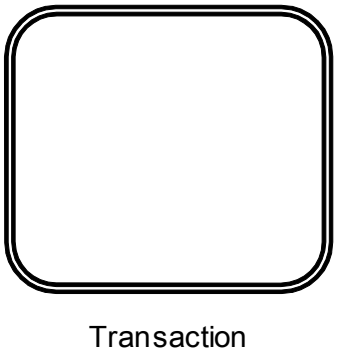


Figure 88 - Transaction Notation

6.5.2.30 Instance: Cancel Event

Class: Behavioral Event

Description

Links

Played End	*Opposite End*
Cancel Event:event part type	*behavioral event usage* Cancel
Cancel Event:induced behavioral event	*behavioral event context* Cancel Process
Cancel Event:packagedElement	*owningPackage* Compensation Library

BPMN Notation

Marker of the **Cancel Event** instance of **Behavioral Event**.

Figure 89 - Behavioral Event 'Cancel' Instance Notation

6.5.2.31 Instance: Cancel Process

Class: Process

Description

6.5.2.32 Links Instance: cancel-end

Class: Happening Succession

Description

Links

Played End	*Opposite End*
cancel-end:next happening succession	*predecessor* Cancel
cancel-end:owned course succession	*owner course* Process Occurrence
cancel-end:previous happening succession	*successor* End

6.5.2.33 Instance: Cancel

Class: Event Part

Description

Links

Played End	*Opposite End*
Cancel:behavioral event usage	*event part type* Cancel Event
Cancel:owned event part	*owner behavior* Process Occurrence
Cancel:predecessor	*next happening succession* cancel-end
Cancel:successor	*previous happening succession* start-cancel

BPMN Notation

Event Part typed by the **Cancel** instance of **Behavioral Event**.

Cancel Event Part

Figure 90 - Event Part : Cancel Notation

6.5.2.34 Instance: Compensate Event

Class: Behavioral Event

Description

Links

Played End	*Opposite End*
Compensate Event:event part type	*behavioral event usage* Compensate
Compensate Event:induced behavioral event	*behavioral event context* Compensate Process
Compensate Event:packagedElement	*owningPackage* Compensation Library

BPMN Notation

Marker of the **Compensate Event** instance of **Behavioral Event**.

Compensate Behavioral Event Instance

Figure 91 - Behavioral Event 'Compensate' Instance Notation

6.5.2.35 Instance: Compensate Process

Class: Process

Description

Links

Played End	*Opposite End*
Compensate Process:behavioral event context	*induced behavioral event* Compensate Event
Compensate Process:packagedElement	*owningPackage* Compensation Library

6.5.2.36 Instance: compensate-end

Class: Happening Succession

Description

Links

Played End	*Opposite End*
compensate-end:next happening succession	*predecessor* Compensate
compensate-end:previous happening succession	*successor* End

6.5.2.37 Instance: Compensate

Class: Event Part

Description

Links

Played End	*Opposite End*
Compensate:behavioral event usage	*event part type* Compensate Event
Compensate:owned event part	*owner behavior* Process Occurrence
Compensate:predecessor	*next happening succession* compensate-end
Compensate:successor	*previous happening succession* start-compensate

6.5.2.38 Instance: Compensation Library

Class: Package

Description

Links

Played End	*Opposite End*
Compensation Library:nestedPackage	*nestingPackage* Activity Library
Compensation Library:owningPackage	*packagedElement* Process Occurrence
Compensation Library:owningPackage	*packagedElement* Cancel Event

Played End	*Opposite End*
Compensation Library:owningPackage	*packagedElement* Compensate Event
Compensation Library:owningPackage	*packagedElement* Compensate Process
Compensation Library:owningPackage	*packagedElement* Cancel Process

6.5.2.39 Instance: Generalization

Class: Generalization

Description

Links

Played End	*Opposite End*
Generalization:	*general* Behavior Occurrence
Generalization:generalization	*specific* Process Occurrence

6.5.2.40 Instance: Process Occurrence

Class: Process

Description

Process based on the **Behavior Occurrence** that produces the following additional lifecycle events: **Compensate Event**, **Cancel Event**.

Links

Played End	*Opposite End*
Process Occurrence:	*owned course succession* startseq-end
Process Occurrence:owner behavior	*owned event part* Compensate
Process Occurrence:owner behavior	*owned event part* Cancel
Process Occurrence:owner course	*owned course succession* cancel-end
Process Occurrence:owner course	*owned course succession* start-compensate
Process Occurrence:owner course	*owned course succession* start-cancel
Process Occurrence:packagedElement	*owningPackage* Compensation Library
Process Occurrence:specific	*generalization* Generalization

6.5.2.41 Instance: start-cancel

Class: Happening Succession

Description

Links

Played End	*Opposite End*
start-cancel:next happening succession	*predecessor* Start
start-cancel:owned course succession	*owner course* Process Occurrence
start-cancel:previous happening succession	*successor* Cancel

6.5.2.42 Instance: start-compensate

Class: Happening Succession

Description

Links

Played End	*Opposite End*
start-compensate:next happening succession	*predecessor* Start
start-compensate:owned course succession	*owner course* Process Occurrence
start-compensate:previous happening succession	*successor* Compensate

6.5.2.43 Instance: StartFromSequence

Class: Event Part

Description

Links

Played End	*Opposite End*
StartFromSequence:predecessor	*next happening succession* startseq-end

6.5.2.44 Instance: startseq-end

Class: Happening Succession

Description

Links

Played End	*Opposite End*
startseq-end:next happening succession	*predecessor* StartFromSequence
startseq-end:owned course succession	Process Occurrence
startseq-end:previous happening succession	*successor* End

6.6 Interaction Protocol Model

6.6.1 Introduction

The Interaction Protocol Model is for capturing choreographies. It enables interactions to be grouped together into larger, reusable interactions. For example, an interaction that exchanges goods between companies might be used with other interactions in a larger protocol representing a partnership of the companies. This protocol might be adopted by a standards body and reused between many pairs of companies. The interactions in a protocol may be simple interactions that have no sub-interactions, or may be other protocols.

The Interaction Model provides:

- Behaviors with steps that are interactions (Interaction Protocols).
- Interactions representing the reuse of protocols (Compound Interactions).
- A way to specify how a reused protocol ties in with the protocols using it (Compound Interaction Binding).

Interaction Protocols are Interactive Behaviors where the Behavior Steps are Interactions between Interaction Roles (see Simple Interaction Model). For example, a protocol between two companies might start with one company sending another an order, then the other sending back a product, and then the original company sending payment, and finally receiving a receipt. These four simple interactions can be grouped into an interaction protocol, with successions between them to specify which interaction comes before which (see the Behavior Model). The two

companies are interaction roles in the protocol (see the Simple Interaction Model).
Compound Interactions are Interactions that are also Behavior Steps, enabling them to reuse Interaction Protocols. For example, two companies might use the ordering protocol described above many times as part of a larger partnership. This partnership is an interaction protocol that reuses the ordering protocol many times. Each reuse is represented as a compound interaction in the larger partnership protocol. Compound Interactions are complementary to Simple Interactions, which are Interactions that do not have sub interactions (see the Simple Interaction Model).

Compound Interaction Bindings are Connected Part Bindings that specify how reused protocols tie in with the larger protocols reusing them (see the Behavior Model). For example, reusing the ordering protocol described above requires specifying which part in the larger partnership identifies the buying company and which identifies the selling company. Both companies will play these roles at some point in the larger partnership, so the bindings must be specified for each compound interaction. Compound Interaction Bindings are also used in processes (see the Activity Model).

6.6.2 Metamodel Specification

The Interaction Protocol Model is for capturing choreographies. It enables interactions to be grouped together into larger, reusable interactions. For example, an interaction that exchanges goods between companies might be used with other interactions within a larger protocol representing a partnership of the companies. This protocol might be adopted by a standards body and reused between many pairs of companies. The interactions in a protocol may be simple interactions that have no sub-interactions, or may be other protocols.

6.6.2.1 Interaction Protocol

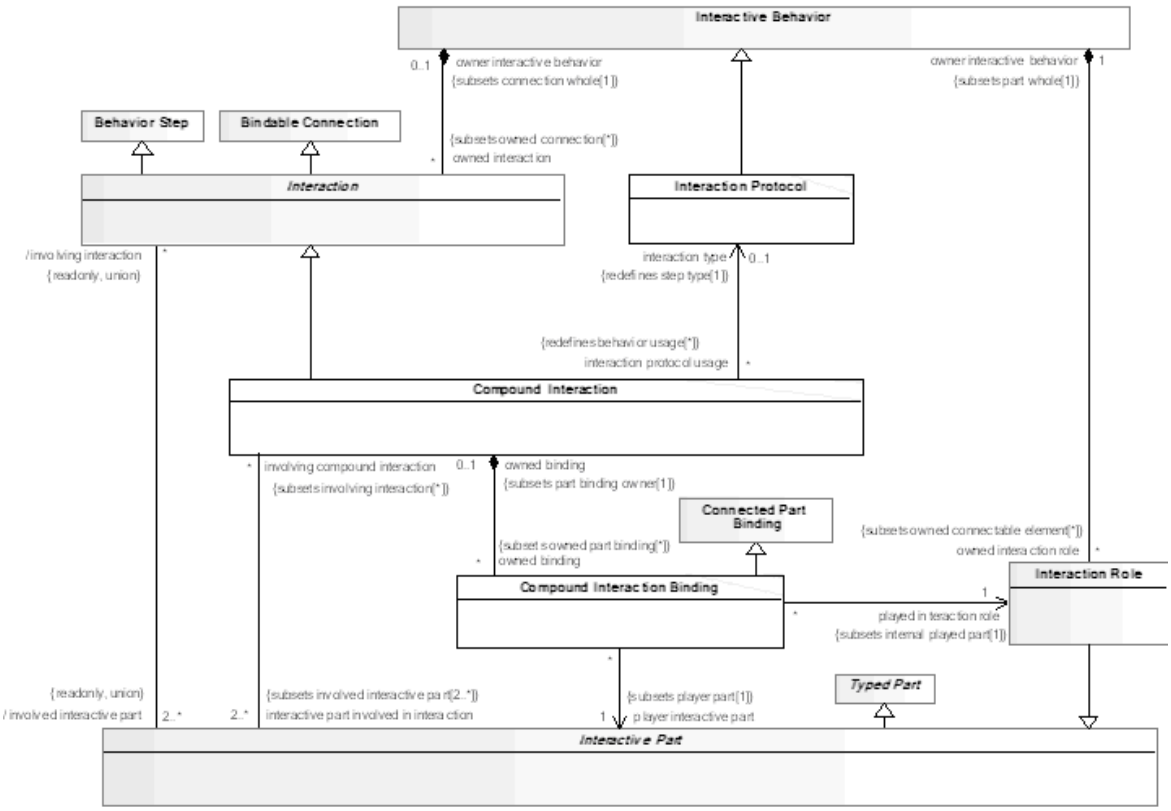


Figure 92 - Interaction Protocol

6.6.2.2 Compound Interaction

Package: Interaction Protocol Model
isAbstract: No
Generalization: “Interaction”

Description

A **Compound Interaction** is an **Interaction** that is also a **Behavior Step**, enabling it to reuse an **Interaction Protocol**. **Compound Interaction** is complementary to **Simple Interaction**, which is an **Interaction** that doesn't have sub-interactions.

Associations

interaction type : Interaction Protocol [0..1]	Interaction Protocol that defines the type of the Compound Interaction.
interactive part involved in interaction : Interactive Part [2..*]	Subsets *involved interactive part*
owned binding : Compound Interaction Binding [*]	Subsets *owned part binding*

Non Normative Notation

A compound interaction is represented by a rounded corner rectangle that MUST be drawn with a double thin black line.

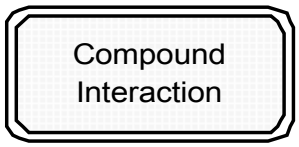


Figure 93 - Compound Interaction Notation

6.6.2.3 Compound Interaction Binding

Package: Interaction Protocol Model
isAbstract: No
Generalization: “Connected Part Binding”

Description

A **Compound Interaction Binding** is a **Connected Part Binding** that specifies how an **Interaction Protocol** reused by a **Compound Interaction** ties in with the larger **Behavior** reusing it. For each **Interactive Part** involved in a **Compound Interaction**, there is a **Compound Interaction Binding** that specifies which **Interaction Role** it plays in the **Interaction Protocol**.

Associations

played interaction role : Interaction Role [1]	The Interaction Role that is played by the player interactive part connected by the Compound Interaction Binding. Subsets *internal played part*
player interactive part : Interactive Part [1]	The Interactive PartInteractive Part being playing the played interaction role as defined by the Compound Interaction Binding. Subsets *player part*

6.6.2.4 Interaction Protocol

Package: Interaction Protocol Model
isAbstract: No

Generalization: “Interactive Behavior”

Description

An **Interaction Protocol** is a kind of **Interactive Behavior** where **Behavior Steps** are **Interactions** that occur between **Interaction Roles**. The set of **Interaction**s defines the purpose of the **Interaction Protocol**.

6.7 Class Hierarchies

The **Class Hierarchies** is not a real package. It groups diagrams that provide a synthesis of class hierarchies.
The **BPDM Class Hierarchies** is not a real package. It groups diagrams that provide a synthesis of class hierarchies.

6.7.1 Condition Hierarchy

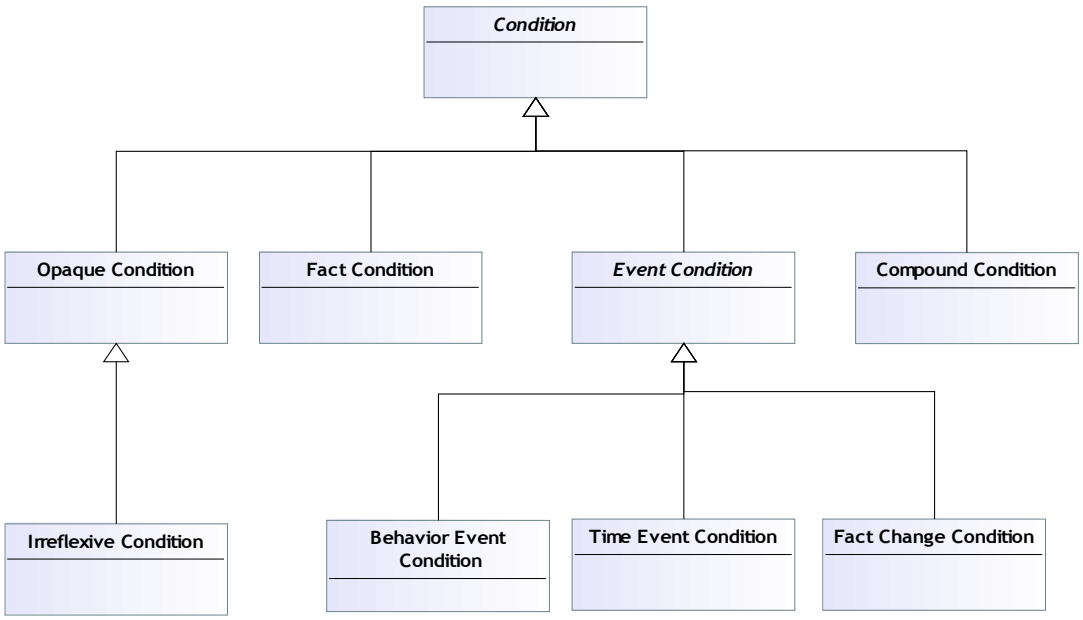


Figure 94 - Condition Hierarchy

6.7.2 Happening OverTime Hierarchy

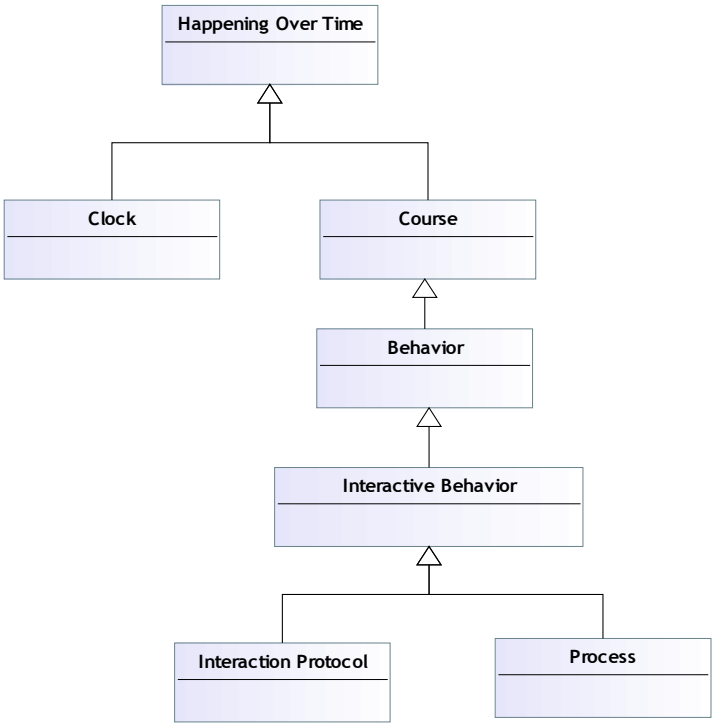


Figure 95 - Happening OverTime Hierarchy

6.7.3 Event Hierarchy

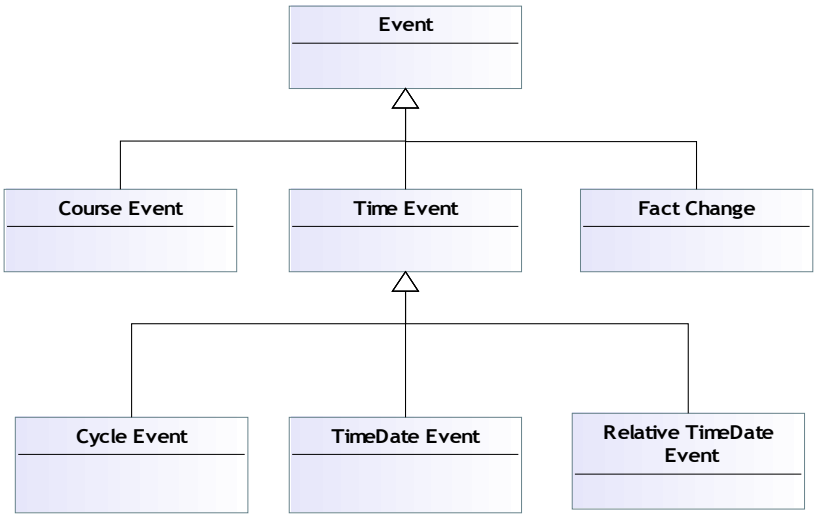


Figure 96 - Event Hierarchy

6.7.4 Behavioral Step Hierarchy

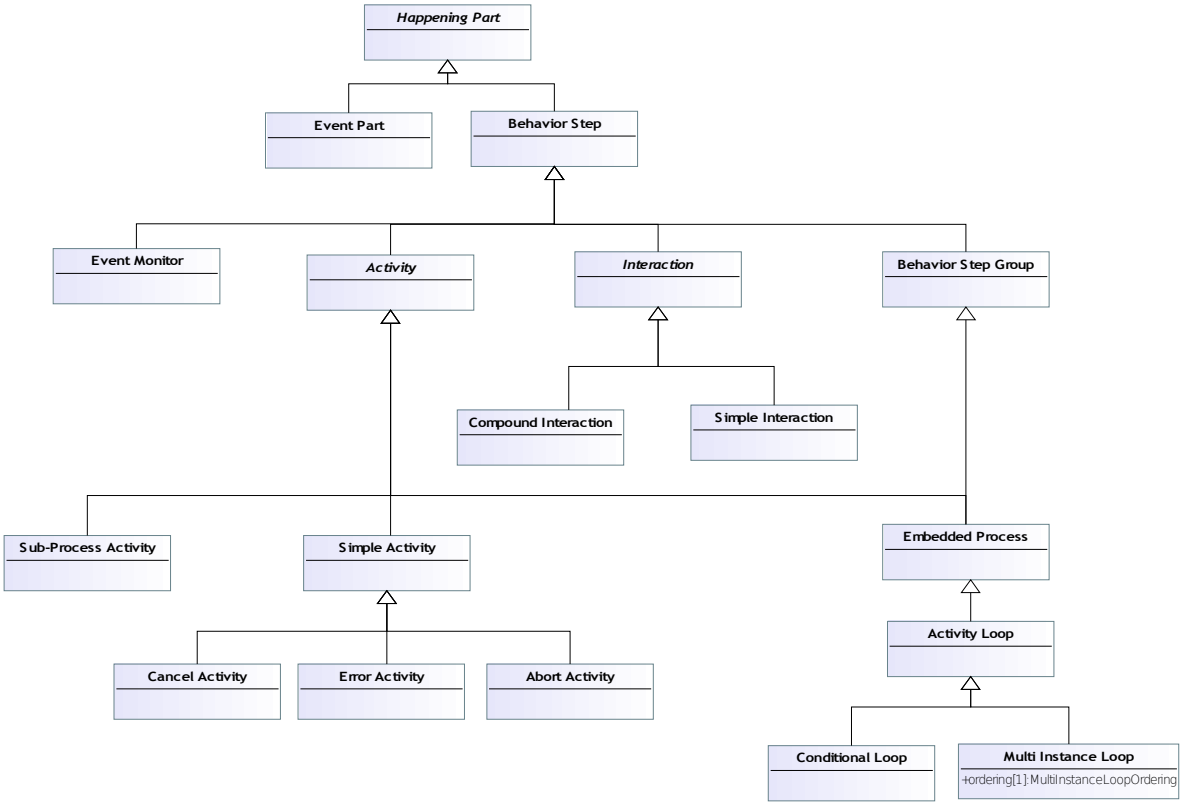


Figure 97 - Behavioral Step Hierarchy

6.7.5 Simple Interaction Hierarchy

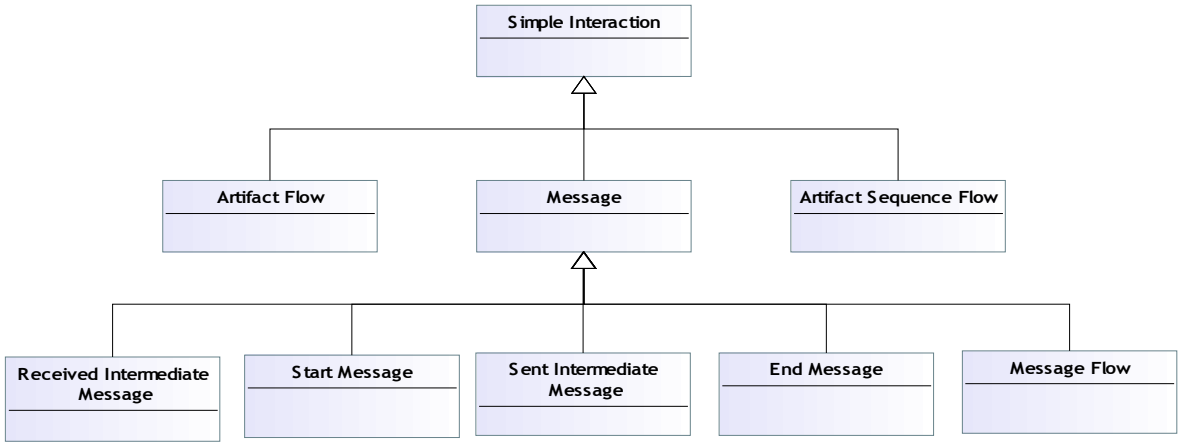


Figure 98 - Simple Interaction Hierarchy

6.7.6 Interactive Part Hierarchy

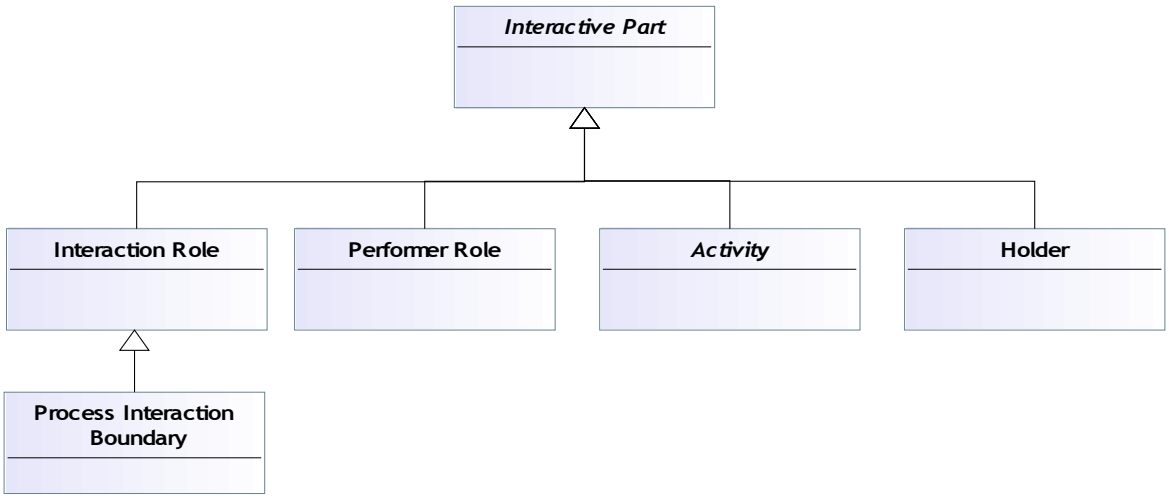
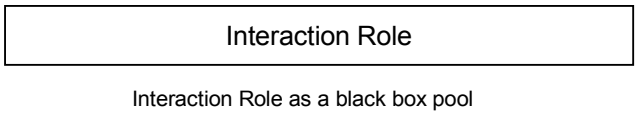


Figure 99 - Interactive Part Hierarchy

7 BPMN Notation Summary

7.1 Interaction Role Notation

A "black box pool" is a pool that does not have any process details.



Interaction Role as a black box pool

Figure 100 - Interaction Role Notation

Represented Elements

Interaction RoleProcess Interaction Boundary

7.2 Processor Role Notation

A Processor Role is represented by a Pool. A Pool is a square-cornered rectangle that MUST be drawn with a solid single black line.

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.

The Processor Role Pool MAY be presented without a boundary.

Figure 101 - Processor Role Notation

Represented Elements

Processor Role

7.3 Horizontal Lane Notation

A Performer Role is represented by a Lane. A lane is a sub-partition of the Pool representing the **Processor Role** of the process or a sub-partition of the Lane representing its delegating performer role.

A Lane will extend the entire length of its containing Pool or Lane, either vertically or horizontally. 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 (the Performer Role name) 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 102 - Horizontal Lane Notation

Represented Elements

Performer Role

7.4 Vertical Lane Notation

A Performer Role is represented by a Lane. A lane is a sub-partition of the Pool representing the Processor Role of the process or a sub-partition of the Lane representing its delegating performer role.

A Lane will extend the entire length of its containing Pool or Lane, either vertically or horizontally. 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 (the Performer Role name) 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.

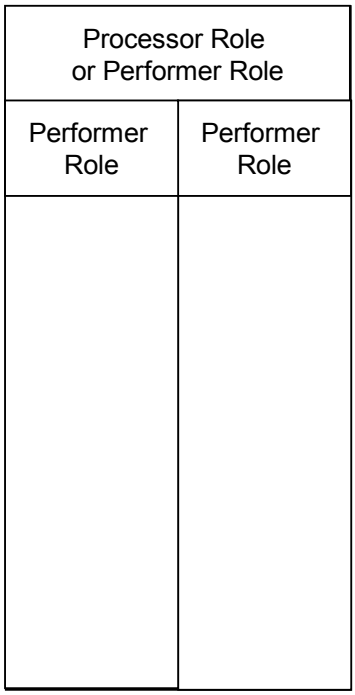


Figure 103 - Vertical Lane Notation

Represented Elements

Performer Role

7.5 Time Event Notation

A **Time Event** is represented by a clock.

Time Event

Figure 104 - Time Event Notation

Represented Elements

Time Event

7.6 Fact Change Notation

Fact Change

Figure 105 - Fact Change Notation

Represented Elements

Fact Change

7.7 Course Event 'Error' Instance Notation

Marker of the **Error Event** instance of **Event**.

Error Behavioral Event Instance

Figure 106 - Course Event 'Error' Instance Notation

Represented Elements

Error Event

7.8 Course Event 'Cancel' Instance Notation

Marker of the **Cancel Event** instance of **Event**.

Cancel Behavioral Event Instance

Figure 107 - Course Event 'Cancel' Instance Notation

Represented Elements

Cancel Event

7.9 Course Event 'Iteration End'

Marker of the **IterationEnd Event** instance of **Event**.

Iteration End Behavioral Event Instance

Figure 108 - Course Event 'Iteration End'

Represented Elements

IterationEnd Event

7.10 Course Event 'Abort' Notation

Marker of the **Abort Event** instance of **Event**.

Abort Behavioral Event Instance

Figure 109 - Course Event 'Abort' Notation

Represented Elements

Abort Event

7.11 Course Event 'Compensate' Instance Notation

Marker of the **Compensate Event** instance of **Event**.

Compensate Behavioral Event Instance

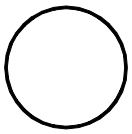
Figure 110 - Course Event 'Compensate' Instance Notation

Represented Elements

Compensate Event

7.12 Event Part : Start Notation

An **Event Part** typed by the **Start Event** instance of **Event** is drawn as a circle that MUST be drawn with a single thin line.



Start Event Part

Figure 111 - Event Part : Start Notation

Represented Elements

Start

7.13 Event Part : Start with 'Time Event Condition' Notation

Shape of **Start** when it has an **Event Monitor** with a **Time Event Condition**, as its predecessor.

Start with Time condition

Figure 112 - Event Part : Start with 'Time Event Condition' Notation

Represented Elements

Start

7.14 Event Part : Start with 'Fact Change Condition' Notation

When a **Start Event Event Part** is conditioned by a **Fact Change Condition**, a **Fact Change** marker is added to the **Start Event Event Part** shape.

Start with Fact Change Condition

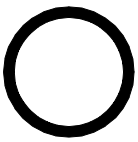
Figure 113 - Event Part : Start with 'Fact Change Condition' Notation

Represented Elements

Start

7.15 Event Part : End Notation

The shape of the **End** instance of **Event Part** is drawn as a circle that MUST be drawn with a single thick black line.



End Event Part

Figure 114 - Event Part : End Notation

Represented Elements

End

7.16 Event Part : Error Notation

The shape of the **Error** instance of **Event Part** use the shape of its super-property (**End**) with the marker of its event type: **Error Event**.

Error Event Part

Figure 115 - Event Part : Error Notation

Represented Elements

Error

7.17 Event Part : Cancel Notation

Event Part typed by the **Cancel** instance of **Event**.

Figure 116 - Event Part : Cancel Notation

Represented Elements

Cancel

7.18 Event Part : Abort Notation

The shape of the **Abort** instance of **Event Part** uses the shape of its super-property (**End**) with the marker of its event type: **Abort Event**.

Figure 117 - Event Part : Abort Notation

Represented Elements

Abort

7.19 Error Handling Notation

Error Event Event Part used for error handling.The **Error Event Event Part** is linked to the **Succession** instance through the **source event part** association.

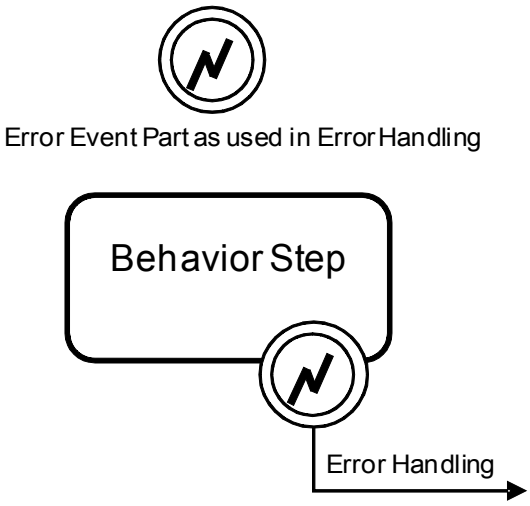


Figure 118 - Error Handling Notation

Represented Elements

Error

7.20 Activity Notation

An Activity is represented by a rounded corner rectangle that MUST be drawn with a single thin black line.

Figure 119 - Activity Notation

Represented Elements

ActivitySimple ActivityTask

7.21 Collapsed Sub-Process Activity Notation

A Sub-Process Activity shares the same shape as the Activity object, which is a rounded rectangle. A Sub-Process Activity is a rounded corner rectangle that MUST be drawn with a single thin black line. If the Sub-Process Activity is also a transaction, it has a boundary drawn with a double line.

The Sub-Process Activity can be in a collapsed view that hides its details or a Sub-Process can be in an expanded view that shows the details of its Process Type.

In the collapsed form, the Sub-Process Activity uses a marker to distinguish it as a Sub-Process Activity, rather than a Simple Activity. The Sub-Process Activity marker MUST be a small square with a plus sign (+) inside. The square

MUST be positioned at the bottom center of the shape.

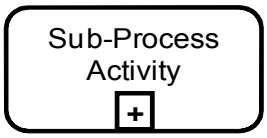


Figure 120 - Collapsed Sub-Process Activity Notation

Represented Elements

Embedded ProcessSub-Process Activity

7.22 Uncollapsed Sub-Process Activity Notation

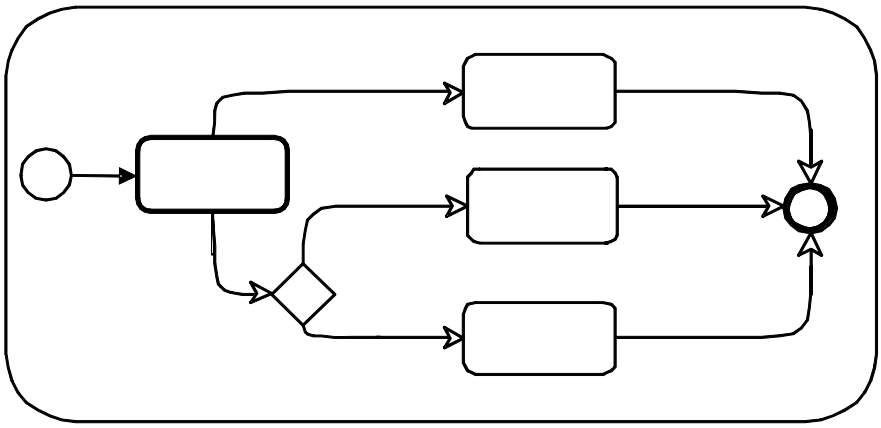


Figure 121 - Uncollapsed Sub-Process Activity Notation

Represented Elements

Embedded ProcessSub-Process Activity

7.23 Activity Loop Notation

An **Activity Loop** has the shape of **Activity** with a small looping indicator that will be displayed at its bottom-center.

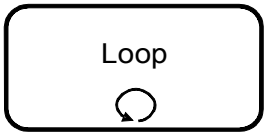


Figure 122 - Activity Loop Notation

Represented Elements

Activity Loop

7.24 Cancel Activity Notation or 'Cancel' Event Part

This symbol can alternatively represent:

1. **Event Part** typed by the **Cancel Event** instance of **Event**.
2. A **Cancel Activity**

Figure 123 - Cancel Activity Notation or 'Cancel' Event Part

Represented Elements

Cancel Activity Cancel

7.25 Error Activity Notation or 'Error' Event Part

This symbol can alternatively represent:

1. **Event Part** typed by the **Error Event** instance of **Event**.
2. An **Error Activity**

Figure 124 - Error Activity Notation or 'Error' Event Part

Represented Elements

Error Activity Error

7.26 Abort Activity Notation or 'Abort' Event Part

This symbol can alternatively represent:

- **Event Part** typed by the **Abort Event** instance of **Event**.
- An **Abort Activity**

Figure 125 - Abort Activity Notation or 'Abort' Event Part

Represented Elements

Abort ActivityTerminate

7.27 Compensate Activity Notation

Figure 126 - Compensate Activity Notation

Represented Elements

Compensate Activity

7.28 Compensating Activity Notation

A **Compensating Activity** shares the standard activity shape with the **Compensate Event** marker displayed in the bottom center of the activity.

Figure 127 - Compensating Activity Notation

Represented Elements

Compensating Activity

7.29 Event Monitor Notation

This symbol is a circle, with an open center. The circle MUST be drawn with a double thin black line. It can alternatively represent:

- **Event Parts** that are not typed by **Start Event** or **End Event**.
- Event Monitors

Markers can be placed within the circle to indicate the nature of the **Event** associated with the **Event Part** or **Event Monitor**.

Figure 128 - Event Monitor Notation

Represented Elements

Event Monitor

7.30 Event Monitor monitoring a Time Event Condition

Event Monitor shape with a **Time Event** as a maker.

Figure 129 - Event Monitor monitoring a Time Event Condition

Represented Elements

Event Monitor

7.31 Event Monitor monitoring a Fact Change Condition

Event Monitor shape with a **Fact Change** as a maker.

Event Monitor for Fact Change

Figure 130 - Event Monitor monitoring a Fact Change Condition

Represented Elements

Event Monitor

7.32 Event Monitor monitoring a 'Compensate' Behavior Event Condition

Event Monitor shape with the marker of the **Compensate Event** instance of **Event**.

Compensation Event Monitor

Figure 131 - Event Monitor monitoring a 'Compensate' Behavior Event Condition

Represented Elements

Event Monitor

7.33 Event Monitor monitoring a Compound Event Condition

Event Monitor
monitoring a Compound Event Condition

Figure 132 - Event Monitor monitoring a Compound Event Condition

Represented Elements

Event Monitor

7.34 Succession Notation

A Succession is a line with a solid arrowhead that MUST be drawn with a solid single line.

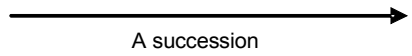


Figure 133 - Succession Notation

Represented Elements

SuccessionSequence Flow

7.35 Event Decision Notation

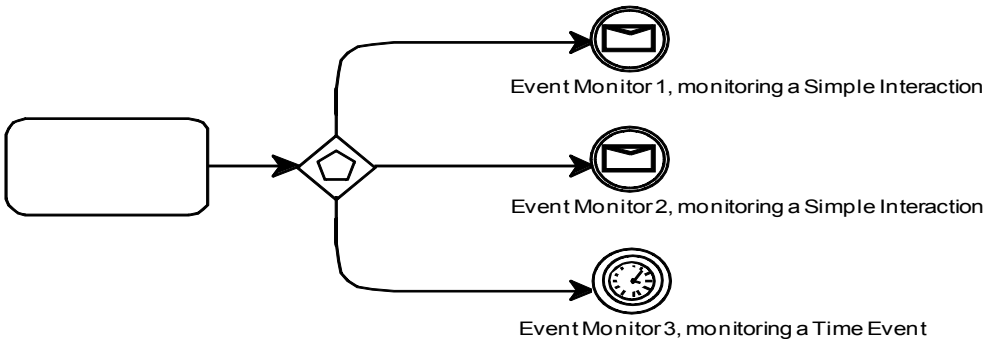


Figure 134 - Event Decision Notation

Represented Elements

Event Decision

7.36 Message Notation

The shape of **Message** depends on its sub-types.

The line connecting a **Message** to its **Interaction Role**(s) MUST have an open arrowhead and MUST be drawn with a dashed single black line.

The line connecting a **Message** to other kind of **Interactive Part** MUST have a solid arrowhead and MUST be drawn with a solid single line.

Figure 135 - Message Notation

Represented Elements

Message

7.37 Start Message Notation

Notation for **Start Message** or **Simple Interaction** categorized as a **Start Message**.

Figure 136 - Start Message Notation

Represented Elements

Start Message

7.38 End Message Notation

Notation for **End Message** or **Simple Interaction** categorized as an **End Message**.

Figure 137 - End Message Notation

Represented Elements

End Message

7.39 Sent Intermediate Message Notation

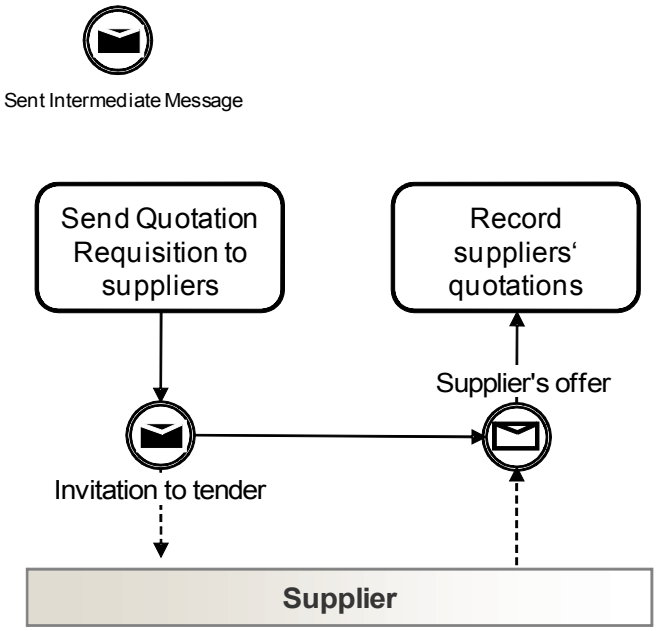


Figure 138 - Sent Intermediate Message Notation

Represented Elements

Sent Intermediate Message

7.40 Received Intermediate Message Notation

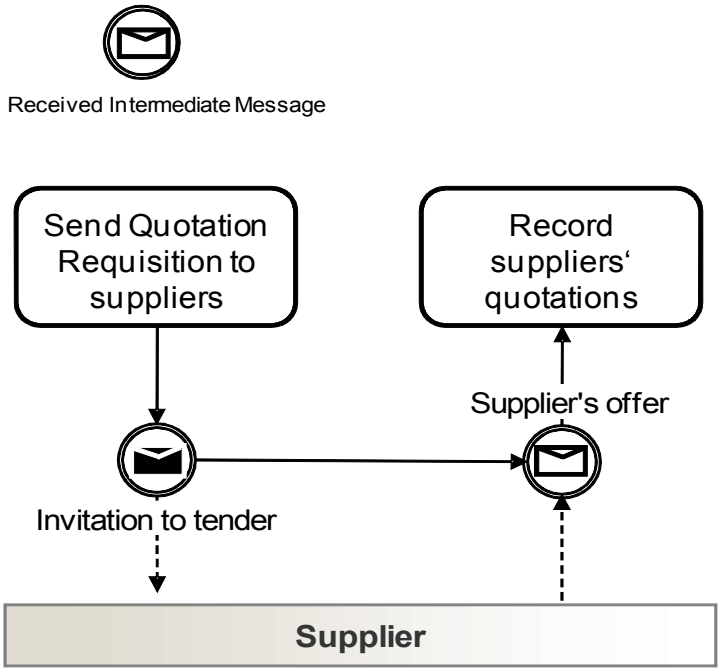


Figure 139 - Received Intermediate Message Notation

Represented Elements

Received Intermediate Message

7.41 Message Flow Notation

A **Message Flow** is a line with an open arrowhead that MUST be drawn with a dashed single black line.

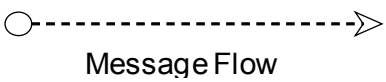


Figure 140 - Message Flow Notation

Represented Elements

Message Flow

7.42 Artifact Sequence Flow Notation

An **Artifact Sequence Flow** is represented by a line with a solid arrowhead that MUST be drawn with a solid single line.

The type of the element transferred by the information flow is represented by a portrait-oriented rectangle that has its upper-right corner folded over that MUST be drawn with a solid single black line.

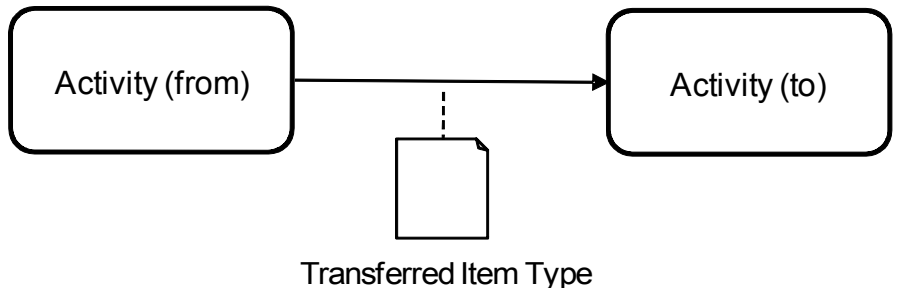


Figure 141 - Artifact Sequence Flow Notation

Represented Elements

Artifact Sequence Flow

7.43 Part Group Notation

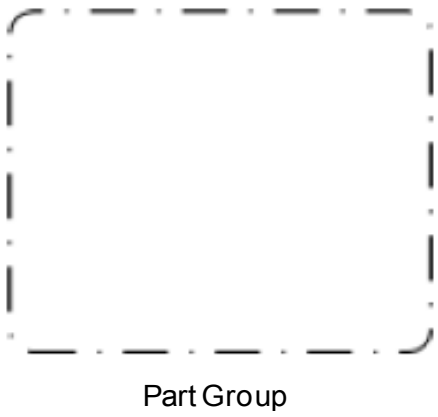


Figure 142 - Part Group Notation

Represented Elements

Part Group

7.44 Transaction Notation

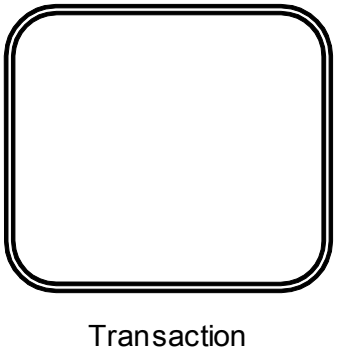


Figure 143 - Transaction Notation

Represented Elements

Transaction

7.45 Gateway Notation

A Gateway is represented by a diamond that has been used in many flow chart notations for exclusive branching and is familiar to most modelers. The diamond MUST be drawn with a single thin black line.

It is not a requirement that predecessor and successor Successions must connect to the corners of the diamond. Successions can connect to any position on the boundary of the Gateway.

The shape of the different sub-types of Gateway are differentiated by an internal marker. This marker MUST be placed inside the shape, in any size or location, depending on the preference of the modeler or modeling tool vendor.

Figure 144 - Gateway Notation

Represented Elements

Gateway

7.46 Exclusive Split Notation

The Exclusive Split shares the same basic shape, called a Gateway, of the generic **Gateway**. The Exclusive Split MAY use a marker that is shaped like an “X” and is placed within the Gateway diamond to distinguish it from other **Gateways**. This marker is not required. A Diagram SHOULD be consistent in the use of the “X” internal indicator. That is, a Diagram SHOULD NOT have some Exclusive Splits with an indicator and some Exclusive Splits without an indicator.

The **default** succession is represented by a default Marker that MUST be a backslash near the beginning of the line representing the **Succession**.

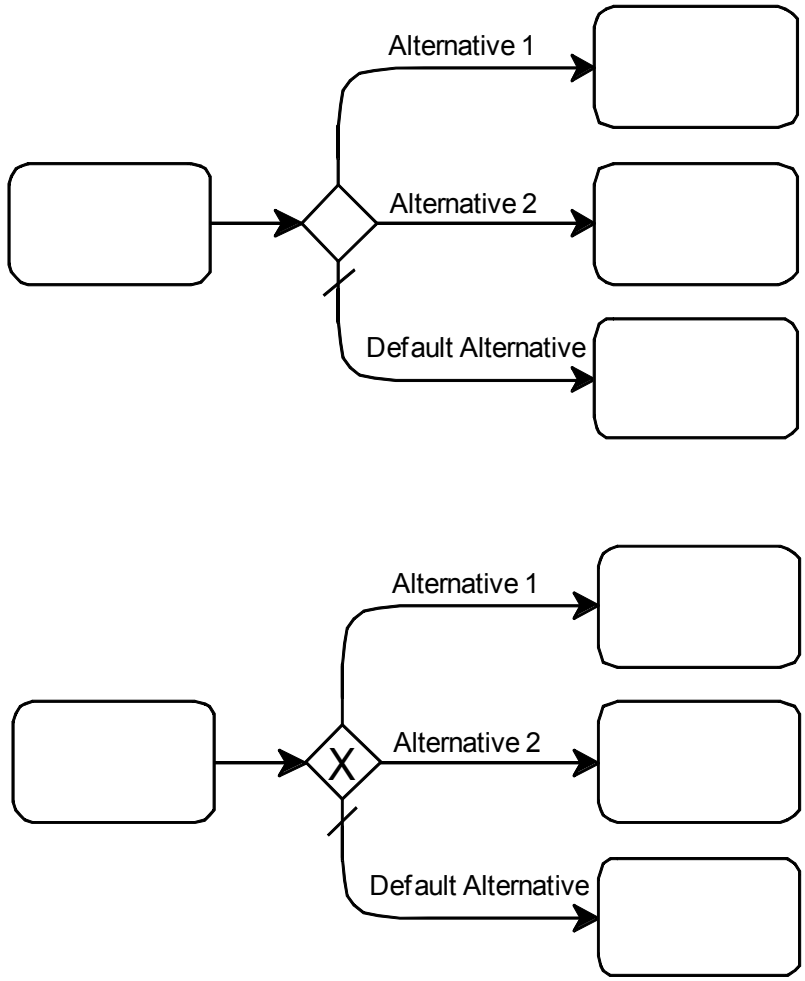


Figure 145 - Exclusive Split Notation

Represented Elements

Exclusive DecisionExclusive Split

7.47 Exclusive Merge Notation

The Exclusive Join shares the same basic shape of the generic **Gateway**.

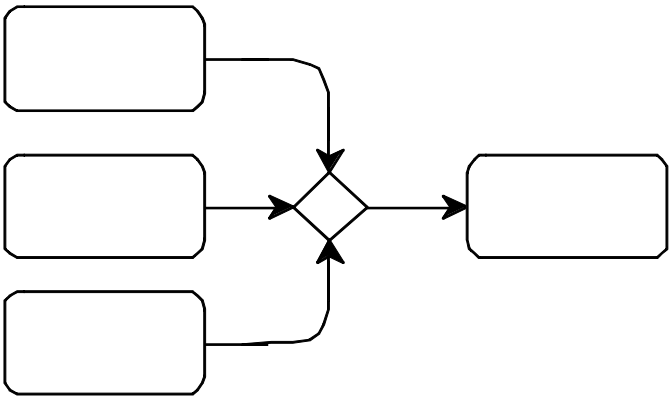


Figure 146 - Exclusive Merge Notation

Represented Elements

Exclusive JoinExclusive Merge

7.48 Parallel Split Notation

The Parallel Split uses the shape of **Gateway**, called Gateway and MUST use a marker that is in the shape of a plus sign and is placed within the Gateway diamond to distinguish it from other of **Gateways**.

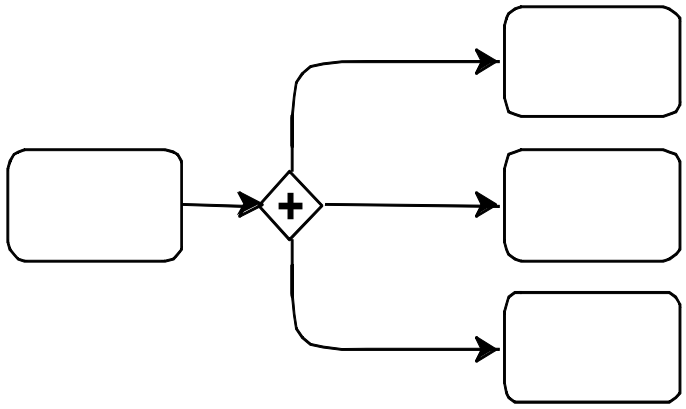


Figure 147 - Parallel Split Notation

Represented Elements

Parallel Split

7.49 Parallel Join Notation

The Parallel Join uses the shape of **Gateway**, called Gateway and MUST use a marker that is in the shape of a plus sign and is placed within the Gateway diamond to distinguish it from other of **Gateways**.

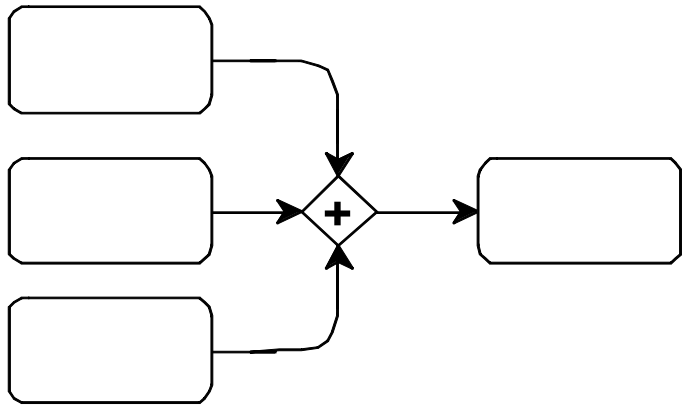


Figure 148 - Parallel Join Notation

Represented Elements

Parallel Join

7.50 Inclusive Split Notation

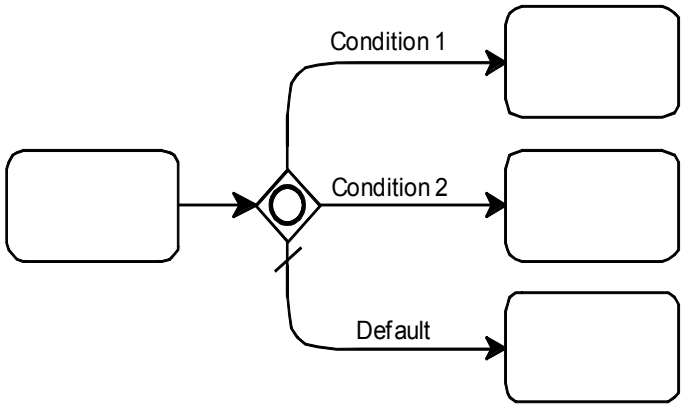


Figure 149 - Inclusive Split Notation

Represented Elements

Inclusive Decision

7.51 Inclusive Merge Notation

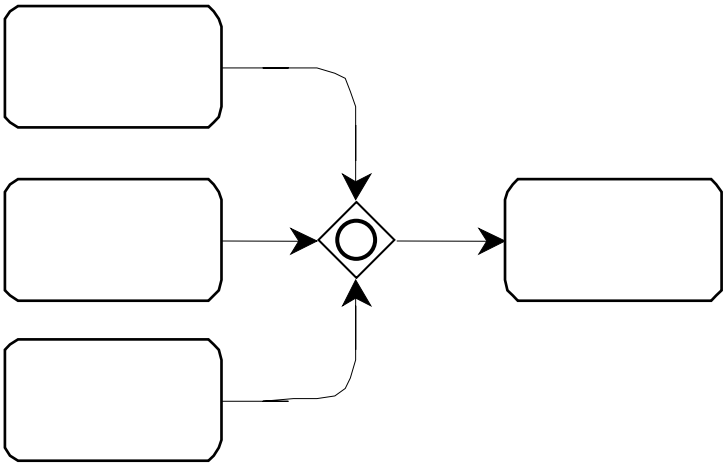


Figure 150 - Inclusive Merge Notation

Represented Elements

Inclusive Merge

7.52 Complex Decision Notation

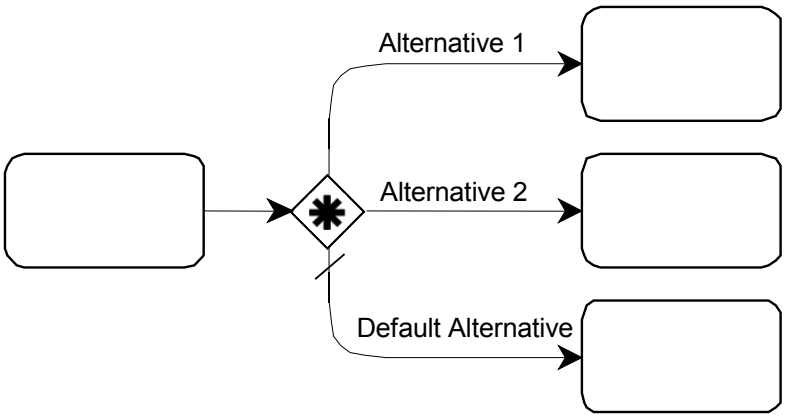


Figure 151 - Complex Decision Notation

Represented Elements

Complex Decision

7.53 Complex Join Notation

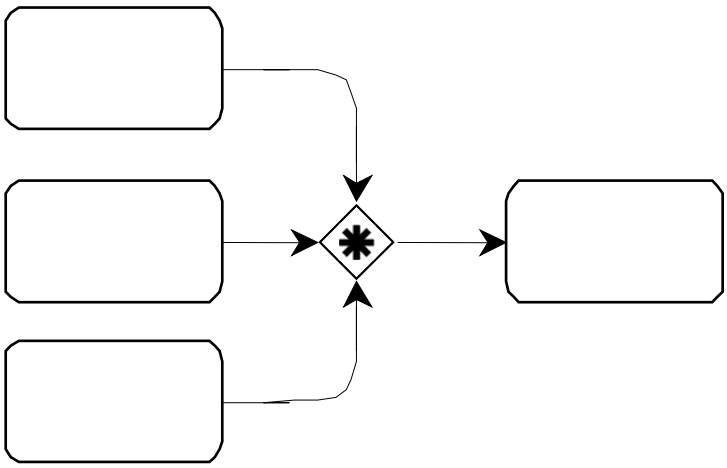


Figure 152 - Complex Join Notation

Represented Elements

Complex Merge

8 Non-normative Notation Summary

8.1 Process Diagram

Each process diagram has a contents area. As an option, it may have a frame and a heading as shown in the following figure. The frame is a rectangle. The frame may be omitted and implied by the border of the diagram area provided by a tool. In case the frame is omitted, the heading is also omitted.

The diagram contents area contains the graphical symbols. The heading is a string contained in name tag (rectangle with cutoff corner) in the upper leftmost corner of the rectangle, with the following syntax: <process name>.

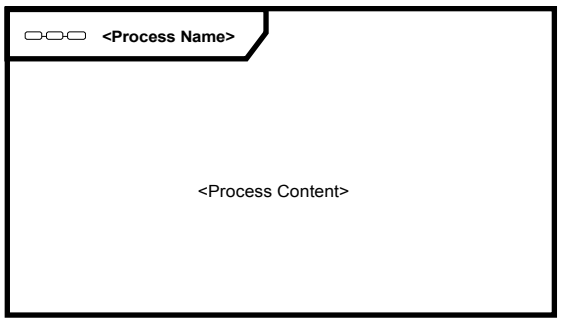


Figure 153 - Process Diagram

Represented Elements

Process

8.2 Non-immediate Succession

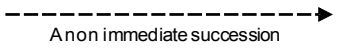


Figure 154 - Non Immediate Succession

Represented Elements

8.3 Course Event 'Normal End' instance notation

Marker of the **Normal End Event** instance of **Event**.

'Normal End' Behavioral Event Instance

Figure 155 - Course Event 'Normal End' instance notation

Represented Elements

Normal End Event

8.4 Course Event 'Abnormal End' instance notation

Marker of the **Normal End** instance of **Event**.

'Abnormal End' Behavioral Event Instance

Figure 156 - Course Event 'Abnormal End' instance notation

Represented Elements

Abnormal End Event

8.5 Course Event 'Failure' Instance notation

Marker of the **Failure Event** instance of **Event**.

Failure Behavioral Change Instance

Figure 157 - Course Event 'Failure' Instance notation

Represented Elements

Failure Event

8.6 Course Event 'Success' Instance Notation

Marker of the **Success Event** instance of **Event**.

Success Behavioral Change Instance

Figure 158 - Course Event 'Success' Instance notation

Represented Elements

Success Event

8.7 Event Part : Normal End Notation

The shape of the **Normal End** instance uses the shape of its super-property (**End**) with the marker of its type: **Normal End Event**.

'Normal End' Event Part

Figure 159 - Event Part : Normal End notation

Represented Elements

Normal End

8.8 Event Part : Abnormal End notation

The shape of the **Abnormal End** instance uses the shape of its super-property (**End**) with marker of its type: **Abnormal End Event**.

'Abnormal End' Event Part

Figure 160 - Event Part : Abnormal End notation

Represented Elements

Abnormal End

8.9 Event Part : Success Notation

The shape of the **Success** instance uses the shape of its super-property (**End**) with marker of its type: **Success Event**.

Figure 161 - Event Part : Success Notation

Represented Elements

Success

8.10 Event Part : Failure Notation

The shape of the **Failure** instance uses the shape of its super-property (**End**) with marker of its type: **Failure Event**.

Figure 162 - Event Part : Failure Notation

Represented Elements

Failure

8.11 Succession with Fact Change Condition

A Succession with a **Condition** of type **Fact Change Condition** is drawn as a line covered by the shape the conditioning **Fact Change**.

The line has a solid arrowhead and MUST be drawn with a solid single line.

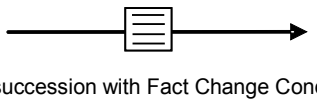


Figure 163 - Succession with Fact Change Condition

Represented Elements

Succession

8.12 Succession with Time Event Condition

A Succession with a **Condition** of type **Time Event Condition** is drawn as one line covered by the shape the conditioning **Time Event**.

The line has a solid arrowhead and MUST be drawn with a solid single line.

A succession with Time Change Condition

Figure 164 - Succession with Time Event Condition

Represented Elements

Succession

8.13 Interaction Flow between Activities and Statement Condition

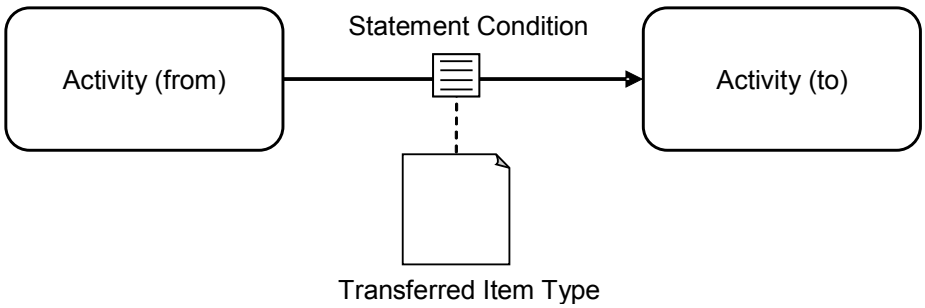


Figure 165 - Interaction Flow between Activities and Statement Condition

Represented Elements

Artifact Sequence Flow

8.14 Interaction Flow between Activities and Time Event Condition

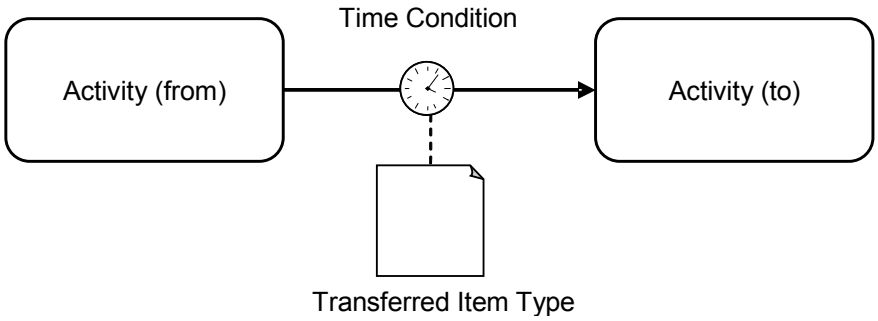


Figure 166 - Interaction Flow between Activities and Time Event Condition

Represented Elements

Artifact Sequence Flow

8.15 Holder Notation

A Holder is represented by a can that MUST be drawn with a single thin black line.

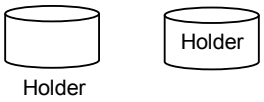


Figure 167 - Holder Notation

Represented Elements

Holder

8.16 Compound Interaction Notation

A compound interaction is represented by a rounded corner rectangle that MUST be drawn with a double thin black line.

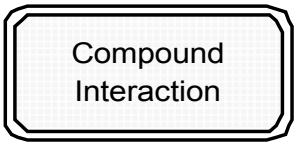


Figure 168 - Compound Interaction Notation

Represented Elements

Compound Interaction

8.17 Course Occurrence Diagram

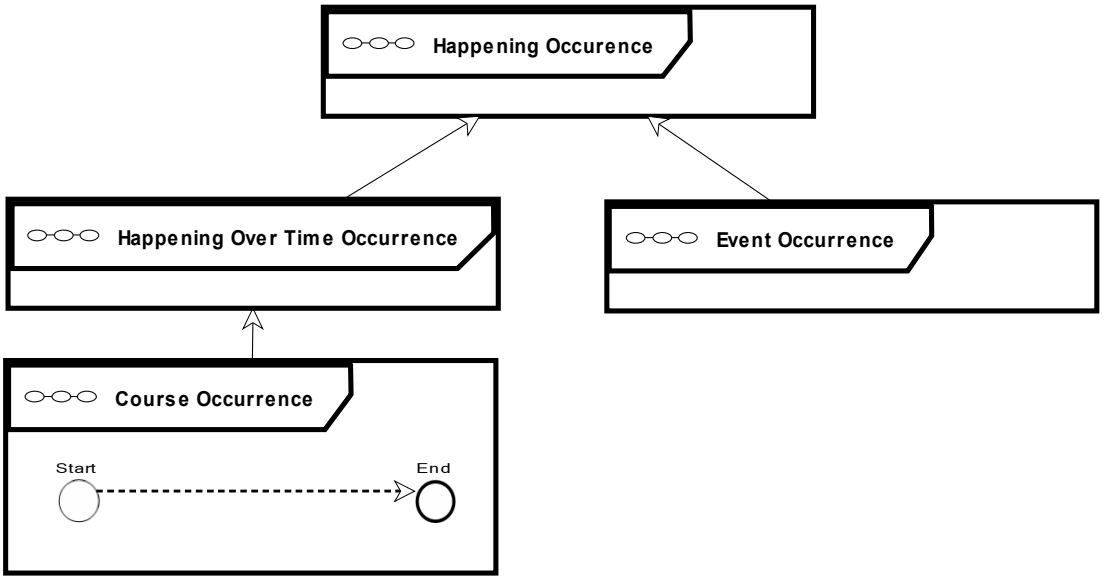


Figure 169 - Course Occurrence Diagram

Represented Elements

Course Occurrence

8.18 Behavior Occurrence

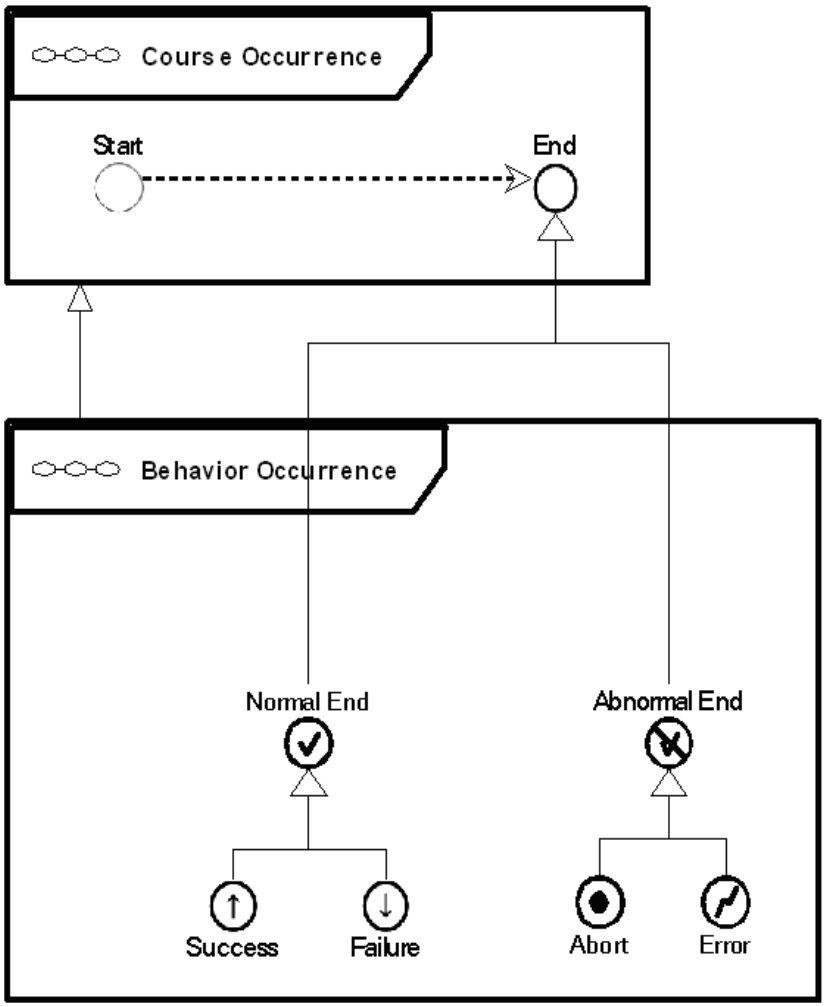


Figure 170 - Behavior Occurrence

Represented Elements

Behavior Occurrence

8.19 Process Occurrence

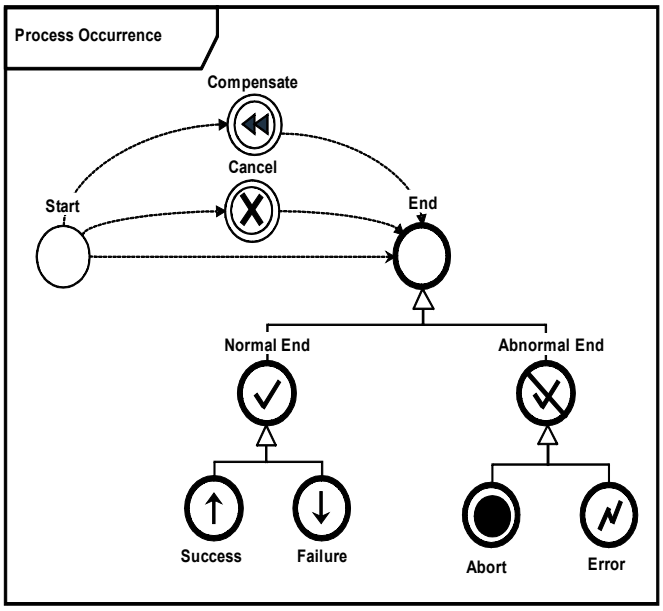


Figure 171 - Process Occurrence

Represented Elements

Process Occurrence

9 BPDM–BPEL Mapping

9.1 General

This section covers a non-normative mapping from BPDM constructs to WS-BPEL 2.0 elements. The basis for the mapping is the “Mapping to BPEL” in [BPMN] (Section 11) and “BPMN to BPDM Mapping” in [BPDM] (Section 6).

9.1.1 Topological Considerations

The Business Process Definition Metamodel (BPDM) is a graph-oriented language in which control and action nodes can be connected almost arbitrarily. In contrast, Business Process Execution Language (BPEL) is a mainly block-structured (albeit providing graph-oriented constructs with syntactical limitations) language with a properly nested structure. As BPDM and BPEL represent two fundamentally different classes of languages, the mapping is technically challenging; while BPEL to BPDM mapping is trivial, not all BPDM processes can easily be converted to BPEL.

To map a BPDM process onto BPEL code, a transformation from a graph structure to block structure is needed. For this purpose, the process can be decomposed into components with one entry and one exit point [BPM-06-02]. These components can then be mapped onto suitable “BPEL blocks.” The decomposition helps to define an iterative approach that allows an incremental transformation of a “componentized” BPDM process to a block-structured BPEL process.

A component may be *well-structured* so that it can be directly mapped onto BPEL structured activities, whereas a non-well-structured component can be translated into BPEL via *event-action rules*. The latter approach can be applied to translating any component to BPEL, yet it produces less readable BPEL code and will therefore be applied only to the remaining non-well-structured components. The algorithm is explained in detail in [BPM-06-02] that addresses the same problem in translation between BPMN and BPEL.

9.1.2 Generator Model

In general transformation from one metamodel to another metamodel requires additional information. This information is provided in a separate model that is specific to the performed transformation. We will refer to this model as "generator model."

If information required by BPEL and not provided by BPDM is needed, then the generator model is responsible for providing it. Such examples are: XML namespaces, specific BPEL customizations, etc. Using the generator model we could avoid introducing concepts and terms in BPDM that are specific for BPEL and still have the capability to customize the produced BPEL models.

Ultimately, a generator metamodel would be required for this generator model in order to describe all possible customizations that can be used. For the purposes of this non-normative mapping, however, it is merely indicated which additional information is needed for the mapping (see Notational Conventions).

9.1.3 Notational Conventions

BPDM constructs are depicted in **Bold** typeface. The equivalent BPMN element may follow in (Parentheses). BPEL elements are represented in <angle brackets> and attributes in *italics*. Marks are denoted in ***Bold Italics***.
The keywords “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

9.2 Process

BPDM	BPEL
Processor Role	**Processor Role** maps to BPEL <process> element. The **NamedElement.name** maps to the *name* attribute of <process>.

9.3 Start Event Mappings

BPDM	BPEL
Event Part typed by the Start Behavioral Event	The only way to instantiate a business process in BPEL is to annotate a <receive> or <pick> activity with the *createInstance* attribute set to "yes." The <receive> or <pick> will likely be placed within a <sequence> or a <flow>.
Start Message	This will map to the <receive> element. The *createInstance* attribute of the <receive> element will be set to "yes." The ***Message*** attribute of **Start** maps to the *variable* attribute of the <receive> element. Note that the extra spaces and non-alphanumeric characters MUST be stripped from the variable attribute to fit with the XML specification of the *variable* attribute. If there is a name collision (because of the name change), then the transformer is responsible for generating unique names. The ***Name*** attribute of **Simple Interaction** maps to the *name* attribute of a BPEL <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 ***Name*** has been stripped. The *messageType*, **t***ype,* or *element* attribute is used to specify the type of a variable. Exactly one of these attributes MUST be used. 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. Attribute *type* refers to an XML Schema type (simple or complex). Attribute *element* refers to an XML Schema element. In case of using a WSDL message type definition, each ***Properties*** 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>. The ***Implementation*** attribute of **Simple Interaction** MUST be a Web service or MUST be converted to a Web service for mapping to BPEL. The Web Service Attributes are mapped as follows: • The ***Participant*** attribute is mapped to the *partnerLink* attribute of the BPEL activity. • The ***Interface*** attribute is mapped to the *portType* attribute of the BPEL activity. • The ***Operation*** attribute is mapped to the *operation* attribute of the BPEL activity.

	• **InteractionFlow.transformationExpression** will map to a <fromParts> element within <receive>.
Time Condition on **Start**	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). During the mapping an additional BPEL process is employed. We will refer to this process as <NameOfStartNode> trigger. Thus the functionality of the timing as defined in the Start Event will be implemented in a separate process that will be started by the BPEL Engine. The process definition will 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. **InteractionFlow.transformationExpression** will map to a <fromParts> element within <receive>.
Fact Change Condition on **Start**	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> element will be mapped as shown for the Message Start Event (see above). **InteractionFlow.transformationExpression** will map to a <fromParts> element within <receive>. Note: The Message is expected to arrive from the application that tracks and triggers Business Rules.

9.4 End Event Mappings

BPDM	BPEL
End Event Part	There is no BPEL element that **Finish** will map to. However, it marks the end of a path within the Process and will be used to define the boundaries of complex BPEL elements.
End Message	This will map to a BPEL <reply> or an <invoke>. The appropriate BPEL 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 has 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>. The ***Message*** attribute of **Finish** maps to the *variable* attribute of the <reply> or the *outputVariable* of the <invoke>. See the corresponding Message Start Event above for more information about how **Simple Interaction** maps to BPEL and WSDL. The ***Implementation*** attribute of **Simple Interaction** MUST be a Web service or MUST be converted to a Web service for mapping to BPEL. The Web Service Attributes are mapped as follows:

	• The ***Participant*** attribute is mapped to the *partnerLink* attribute of the BPEL activity. • The ***Interface*** attribute is mapped to the *portType* attribute of the BPEL activity. • The ***Operation*** attribute is mapped to the *operation* attribute of the BPEL activity. **InteractionFlow.transformationExpression** will map to the *fromVariable* variable of <toParts> element within <reply> or <invoke>.
Error Activity	This will map to a <throw> element. The ***ErrorCode*** attribute of **Error Activity** will map to the *faultName* attribute of the <throw>.
Cancel Activity	The mapping of Cancel Activity to BPEL is an open issue.
Abort Activity	This will map to the <exit> element.

9.5 Intermediate Events

BPDM	BPEL
Simple Interaction coming from or going to the **Process Interaction Boundary** that is not connected to **Start** or **Finish**	If **Simple Interaction.Simple Interaction consumer** refers to 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 **Simple Interaction.Simple Interaction producer** is the same Participant as that of the Process that contains the Event, then this will map to a (one-way) <invoke>. The ***Message*** attribute of the Event maps to the *variable* attribute of the <receive> or the *outputVariable* of the <invoke>. See the corresponding Start event above for more information about how **Simple Interaction** maps to BPEL and WSDL. The ***Implementation*** attribute of **Simple Interaction** MUST be a Web service or MUST be converted to a Web service for mapping to BPEL. The Web Service Attributes are mapped as follows: • The ***Participant*** attribute is mapped to the *partnerLink* attribute of the BPEL activity. • The ***Interface*** attribute is mapped to the *portType* attribute of the BPEL activity. • The ***Operation*** attribute is mapped to the *operation* attribute of the BPEL activity. If the Event has no incoming **Processing Succession**: • **Simple Interaction.Simple Interaction consumer** MUST be the same Participant as that of the Process that contains the Event.

	• The <process> could be given a <scope> (if it doesn't already have one). • An <eventHandlers> element can be defined directly under <process> or under <scope> (if one was generated). • An <onMessage> element will be added to the <eventHandlers> element. • The ***Message*** attribute of the Event maps to the *variable* attribute of the <onMessage>. Further, the ***Implementation*** attribute of **Simple Interaction** MUST be a Web service or MUST be converted to a Web service for mapping to BPEL. 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 <onMessage>.
Processing Succession from the abort Event Part	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> above. The activity for the <onMessage> will be a <throw> with "<message name>_Exit" as the *faultName* attribute. If 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.
Time Event Condition on **Succession**	This will map to a <wait>. **TimeEvent.timeExpression** maps to the *until* attribute of the <wait>. **Cycle Event.timeExpression** maps to the *for* attribute of the <wait>. If the Event has no incoming **Processing Succession**: • The <process> could be given a <scope> (if it doesn't already have one). • An <eventHandlers> element will be defined for the process or the <scope> (if <scope> element was generated). • An <onAlarm> element will be added to the <eventHandlers> element. • **TimeEvent.timeExpression** maps to the *until* attribute of the <onAlarm>. • **Cycle Event.timeExpression** maps to the *for* attribute of the <onAlarm>.

Racing Connection connecting an **Event Monitor** conditioned by a **Time Event Condition**	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 "<Event name>_Exit" as the *faultName* attribute. An <eventHandlers> element will be defined for the scope. The Event will map to an <onAlarm> element within the <eventHandlers>. **TimeEvent.timeExpression** maps to the *until* attribute of the <onAlarm>. **Cycle Event.timeExpression** 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. If used in an event-based decision, this will map to an <onAlarm> within a <pick>. **TimeEvent.timeExpression** then maps to the *until* attribute of the <onAlarm>. Accordingly, **Cycle Event.timeExpression** maps to the *for* attribute of the <onAlarm>.
Processing Succession from the errorPart	Within the normal flow, **Processing Succession** will map to a <throw> element. • If the error is attached to an activity, 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 Behavioral Event does not have an ***ErrorCode***, then a <catchAll> element will be added to the <faultHandlers> element. • If the Error Behavioral Event has an ***ErrorCode***, then a <catch> element will be added to the <faultHandlers> element with the ***ErrorCode*** mapping to the *faultName* attribute.
Processing Succession from the abortPart	The mapping of succession from abort to BPEL is an open issue.
Fact Change Condition on **Succession**	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). If the Event has no incoming **Processing Succession**: • Simple Interaction.Simple Interaction consumer MUST be the same Participant as that of the Process that contains the Event.

	• The <process> could be given a <scope> (if it doesn't already have one). • An <eventHandlers> element will be defined for the process or the <scope> (if one was generated). • 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.
Racing Connection connecting **a Event Monitor** monitoring **a Fact Change Condition**	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. If used in an event-based decision, this will map to an <onMessage> element within <pick>. The mapping to the <onMessage> attributes is the same as described for the <receive> for the Message Event above.
Event Monitor monitoring a **Compensation Event**	Within the normal flow: Maps to a <compensate> or <compensateScope> element. The ***Name*** of the activity referenced by the Compensation Event will map to the *target* attribute of the <compensateScope> element. Attached to an activity boundary: The activity (to which the Event is attached) will be placed within a <scope>. This Event maps to a <compensationHandler> element within a <scope>. For the <invoke> activity, there is a special shortcut to inline a <compensationHandler> within <invoke> rather than explicitly using an immediately enclosing scope.

9.6 Activities

BPDM	BPEL
Simple Activity	An incoming **Simple Interaction** maps to a <receive> activity. The ***Message*** attribute maps to the *variable* attribute of the <receive> activity. If the **Simple Interaction** represents start Simple Interaction, then the createInstance attribute of the receive will be set to "yes." Two **Simple Interaction**s associated with the same activity: • An incoming and an outgoing flow • Map to an <invoke> activity. The ***InMessage*** attribute maps to the *inputVariable* attribute of the <invoke> activity. The ***OutMessage*** attribute maps to the *outputVariable* attribute. An outgoing **Simple Interaction** maps to a <reply> or an <invoke> activity. The appropriate BPEL activity will be determined by checking if an upstream <receive> has the same *portType* and *operation*. If these two attributes are matched, then the activity will map to a <reply>, if not, it will map to an <invoke>. The ***Message*** attribute maps to the *variable* attribute of the <reply> activity or it maps to the *inputVariable* attribute of the <invoke> activity. See the Start event above for more information about how **Simple Interaction** maps to BPEL and WSDL.
Script Activity	This 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 is invoked.
Embedded Process	This will map to a <scope> element. The scope is not an independent <process> and will share the process variables of the higher-level process.
Sub-Process Activity	BPEL does not have a sub-process element. Thus Independent Sub-Processes MUST map to a BPEL <process>; the contents of the Sub-Process will be contained within a separate process. The Sub-Process object itself maps to an <invoke> activity that "calls" the process. BPEL does not support Reference type of Sub-Processes. However, the Sub-Process will be used as a placeholder for the Sub-Process that will be mapped. Mapping for an Independent Sub-Process: • The ***DiagramRef*** and ***ProcessRef*** attributes will identify the process that will be used for the mapping to the BPEL process. • The ***OutputPropertyMaps*** attribute of the referenced process maps to the *inputVariable* attribute of the <invoke> activity. • The ***InputPropertyMaps*** attribute of the referenced process maps to the *outputVariable* attribute of the <invoke> activity.

	See the Start event above for more information about how **Simple Interaction** maps to BPEL and WSDL. Mapping for a Reference Sub-Process: • The ***SubProcessRef*** 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; another copy of the entire mapping will be created in this location (the mappings will also exist in the referenced Sub-Process' original location).
Course Control Part	**Course Control Part** will map to a variety of BPEL elements (e.g., <if>, <pick>, <flow>) and patterns of elements. **Course Control Part** potentially marks the end of a BPEL structured element, if the correct number of flows converge. The elements that follow **Course Control Part**, until all the outgoing paths have converged, will be contained within the extent of the mapping (e.g., they will be placed within a <sequence> within an <if><condition> and any number of <if><elseif><condition>s).
Exclusive Split **Exclusive Join**	**Exclusive Split** will map to <if>. Each ***Gate*** will map to branches specified by <if> and <elseif> (within <if>). The order of branches is maintained. Each **guard** association between **Succession** and **Condition** associated with the Gates will map to <condition> elements within <if> or <elseif>. The Default Gate (**ExclusiveSplit.default**) will map to the <else> element of <if>. If there is more than one element that follows the Gate or the Default Gate, including assignments, then these elements will be placed inside a <sequence>.
Embedded Process with an **Event Monitor connected by a Racing Connection**	This will map to <pick>. The elements of the <pick> will be determined by the targets of the outgoing **Processing Succession**. • If the ***Instantiate*** attribute is set to False, the *createInstance* attribute of the <pick> MUST NOT be included OR it MUST be set to "no." • If the ***Instantiate*** attribute is set to True, the *createInstance* attribute of the <pick> MUST NOT be included OR it MUST be set to "yes." • If the target is a **Simple Activity** with an incoming **Simple Interaction**, it maps to an <onMessage> element within <pick>. • The attributes of the **Simple Activity** will map to the appropriate elements of the <onMessage>, such as *operation* and *portType*. • If there is a **Time Event Condition** on **Succession**, the activity maps to an <onAlarm> element within <pick>. • **TimeEvent.timeExpression** maps to the *until* attribute of the <onAlarm>. • **Cycle Event.**timeExpression maps to the *for* attribute of the <onAlarm>. • If there is a **Fact Event Condition** on **Succession**, the event will be considered as the same as receiving a message from a system

	that tracks and generates Rule events. Thus, this will map to an <onMessage> element within the <pick>. • If there is more than one element that follows the first target, including assignments, then these elements will be placed inside a <sequence> activity.
Parallel Split **Parallel Join**	This will map to <flow>.

Inclusive Split
Inclusive Join
	Inclusive Split will map to a set of <if>s within a <flow>. An additional <if> will be required if the Default Gate (**InclusiveSplit.default**) is used.
	
	Each ***Gate*** will map to <if>, which is binary in nature, i.e., has only the main <if> branch and the <else>.
	
	Each **guard** association between **Succession** and **Condition** associated with the Gates will map to <condition> elements within <if> or <elseif>.
	
	• If the Default Gate is used, the mapping to BPEL is more complicated, as the decision about whether the Default Gate should be taken will occur after all the other gate decisions have been determined. Only if no other path is taken, will the default path be taken. This means that the <if> for the Default Gate will follow the <flow> activity generated for all the Gates of the Gateway. Also, a <sequence> activity must encompass the <flow> and the <if>.
	• If the Default Gate is not used, the <else> element for each <if> will contain an <empty> activity.
	
	A <variable> must be used so that the <if> for the Default Gate will know whether or not the default path should be taken. To do this, a BPEL <variable> must be created with a derived name and will have a structure as follows:
	
	`<variable name="[Gateway.Name]_noDefaultRequired"`
	`  messageType="noDefaultRequired" />`
	
	The *messageType*, **t***ype* or *element* attribute is used to specify the type of a variable. Exactly one of these attributes MUST be used. 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. Attribute *type* refers to an XML Schema type (simple or complex). Attribute *element* refers to an XML Schema element.
	
	If a WSDL <message> element is created to support this variable, the message will be structured as follows:
	
	`<message name="noDefaultRequired" >`
	`  <part name="noDefault" type="xsd:boolean" />`
	`</message>`
	
	An <assign> activity will be created to initialize the <variable> before the start of the loop. This <assign> precedes the <flow> activity that contains all the <if>s derived from the Gates. This will be the first activity within the <sequence> activity.
	
	The <assign> will be structured as follows:

	<assign name="[Gateway.Name]_initialize_noDefault"> <copy> <from>false</from> <to variable="[Gateway.Name]_noDefaultRequired" part="noDefault" /> </copy> </assign> The <condition> for the <if> will use the *noDefaultRequired* variable and will be structured as follows: <if> <condition> bpel:getVariableProperty([Gateway.Name]_noDefaultRequired, noDefault)=true </condition> <sequence> <!--The mappings of the original activity are placed here.--> <!--An assign activity (see below) is placed here.--> </sequence> <else> <empty/> </else> </if> • If there is more than one element that follows the first target, including assignments, then these elements will be placed inside a <sequence> activity. • If any of the <if>s within the <flow> passes the condition of the <if>, then the *noDefaultRequired* must be set to True. This will ensure that the Default Gate will bypass the mapped activities for the 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: <assign name="[Gateway.Name]_set_noDefault"> <copy> <from>true</from> <to variable="[Gateway.Name]_noDefaultRequired" part="noDefault" /> </copy> </assign>
Complex Split **Complex Join**	N/A

9.7 Flows

BPDM	BPEL
Processing Succession	This MAY map to a <link> element. In many situations, however, **Processing Succession** will not map to a <link> element; to connect activities that are listed in a BPEL structured activity (e.g., a <sequence>), the <link> elements are not required. <link> elements are only appropriate when the **Processing Successions** are Connecting Objects within a <flow>. However, only the **Processing Successions** that are completely contained within the boundaries of the <flow> will be mapped to a <link>. If another structured activity (e.g., a <while>) is contained within the flow, then the **Processing Successions** that would be appropriate for the contents of the structured activity, would not be mapped to a <link>. If the **Processing Succession** is being mapped to a <link>: • The Name attribute of the Process (**NamedElement.name**) 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. • The mapping of the source activity will include a <source> element. • The Name of the **Processing Succession** (**NamedElement.name**) 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 *linkName* attribute. If the source object is a **Course Control Part** and it maps to an activity, the mapping is the same as if the source object is an activity (see above). If the **Course Control Part** does not map to an activity, the **Processing Succession** will be combined with one of the **Course Control Part**'s incoming **Processing Succession**s. (There will be a separate <link> for each of the incoming **Processing Succession**s). The source of the second **Processing Succession** will be used at the source for the original **Processing Succession**. Then this mapping is the same as if the source object is an activity (see above). The mapping of the target activity will include a <target> element. The Name of the **Processing Succession** (**NamedElement.name**) 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 *linkName* attribute. If the target object is a Gateway and it maps to an activity, the mapping is the same as if the target object is an activity (see above). If the **Control Course Part** does not map to an activity, the **Processing Succession** will be combined with one of the **Course Control Part**'s outgoing **Processing Succession**s. (There will be a separate <link> for each of the outgoing **Processing Succession**s).

	The target of the second **Processing Succession** will be used at the target for the original **Processing Succession**. Then this mapping is the same as if the target object is an activity (see above).
Processing Succession with **Condition**	A <flow> will be required and the **Processing Succession** will map to a <link> element. An <empty> activity will be placed in the flow and will contain all the <source> elements. The **Condition** will then map to the *transitionCondition* attribute of the <source> element that is contained in the appropriate BPEL activity. The mapping of **Processing Succession** with **Condition** when the source object is a **Course Control Part** is described in **Exclusive Split/ Join** and **Inclusive Split/Join**.
ExclusiveSplit.default **InclusiveSplit.default**	See **Exclusive Split/Join** and **Inclusive Split/Join**.

Processing Succession from the **errorPart Event Part**	All the activities that follow the **Processing Succession**, until the Exception Flow merges back into the Normal Flow, will be mapped to BPEL and then placed within the <faultHandlers> element for the <scope> of the activity (and usually within a <sequence>).
	
	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).
	
	The mapping of the original activity will be placed within a <sequence> (if it had not been already). The original activity will be followed by an <if>, with one <condition> and an empty <else> as follows:
	
	`<if>`
	`  <condition>`
	`  bpel:getVariableProperty(`
	`  [activity.Name]_normalCompletion,`
	`  normalCompletion)=true`
	`  </condition>`
	`  <sequence>`
	`<!--The mappings of the Process activities until the merging of the Exception Flow are placed here.-->`
	`  </sequence>`
	`  <else>`
	`  <empty/>`
	`  </else>`
	`</if>`
	
	A <variable> must be used so that the <if> will know whether or not the Exception Flow or Normal Flow had reached that point in the Process. It must be created with a derived name and will have structure as follows:
	
	`<variable name="[activity.Name]_normalCompletion"`
	`  messageType="noDefaultRequired" />`
	
	The *messageType*, t*ype* or *element* attribute is used to specify the type of a variable. Exactly one of these attributes MUST be used. 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. Attribute *type* refers to an XML Schema type (simple or complex). Attribute *element* refers to an XML Schema element.

	If a WSDL <message> element is created to support this <variable>, the message will be structured as follows: `<message name="noDefaultRequired" >   <part name="normalCompletion" type="xsd:boolean" /> </message>` The <assign> will be created to initialize the <variable> before the start of the original activity. The <assign> will be structured as follows: `<assign name="[activity.Name]_initialize_normalCompletion">   <copy>   <from>true</from>   <to variable="[activity.Name]_normalCompletion" part="normalCompletion" />   </copy> </assign>` 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>false</from>   <to variable="[activity.Name]_normalCompletion" part="normalCompletion" />   </copy> </assign>`
Simple Interaction	No specific mapping to a BPEL element. It represents a message that is sent through a WSDL operation that is referenced in a BPEL <receive>, <reply>, or <invoke>. See Start, Intermediate, and End Events for mappings pertaining to **Simple Interaction**.
Event Monitor monitoring **Compensation**	See **Compensation Connection** in Intermediate Events.

9.8 Additional Constructs

BPDM	BPEL
Activity with **Conditional Loop**	This will map to a <forEach> activity. The <forEach> iterates its child <scope> activity exactly N+1 times where N equals the <finalCounterValue> minus the <startCounterValue>.
Activity with **For Loop** or **Multi Instance Loop**	A Multi Instance Loop can be either sequential or parallel. **MultiInstanceLoop.ordering** maps to the *parallel* (="yes|no") attribute of <forEach>. A sequential MI loop maps to <forEach> as in Basic Loop above so that forEachCount equals to N + 1.

	Four flow conditions (None | One | All | Complex) exist for parallel multi-instance loops: • None – There is no synchronization or control of the Tokens that are generated through the multi-instance activity. Each Token will continue on independently and each Token will create a separate instantiation of each activity they encounter. Basically, there is a separate copy of the whole process, for each copy of the MI activity, after that point. Each copy of the remainder of the process will continue independently. • One – Only one of the spawned processes must be completed before the original process can continue. • All – All of the spawned processes must be completed before the original process can continue. • Complex – The difference from All is that the number of completed spawned processes required before the process flow will continue must be determined and the processes have been completed. The BPDM **Activity Loop** is kind of **Embedded Process** that can execute its content multiple times. Upon completion of each iteration the activity loop will generate **Iteration Finish** event. This event can be used in the outgoing **Succession**s to specify that a Succession should be activated on loop iteration completion. Depending on the flow condition: • None – **Succession** on **Iteration Finish** of **Activity Loop** • One – **Succession** on **Iteration Finish** of **Activity Loop** with **Succession.guard** evaluating to the string "first iteration only" • All – **Succession** on **Finish** of **Activity Loop** • Complex – on **Iteration Finish** of **Activity Loop** with **Succession.guard** evaluating to a boolean value. If the value is True ,then the Succession will be followed. A <completionCondition> may be used within the <forEach> to allow the <forEach> activity to complete without executing or finishing all the branches specified. The <forEach> activity without a <completionCondition> completes when all of its child <scope>s have completed. The <completionCondition> element is optionally specified to prevent some of the children from executing (in the serial case), or to force early termination of some of the children (in the parallel case). The <branches> element within <completionCondition> represents an unsigned –integer expression used to define a completion condition of the "at least N out of M" form. The actual value B of the expression is calculated once, at the beginning of the <forEach> activity. It will not change as the result of the <forEach> activity's execution. At the end of execution of each directly enclosed <scope> activity, the number of completed children is compared to B, the value of the <branches> expression. If at least B children have completed, the <completionCondition> is triggered: No further children will be started, and currently running children will be terminated. The mapping to BPEL per flow condition is as follows:

	• None – This is not supported by <forEach>. To create this behavior, the remainder of the process will be moved into a new derived <process>. This process will be spawned through a one-way <invoke> that will be placed within the <while> activity. • One – <completionCondition> evaluates to 1. • All – No <completionCondition> specified. • Complex – <completionCondition> evaluates to B (1 < B < N + 1).

Holder	A BPDM Process can define multiple **Holder** objects. A BPDM **Holder** specializes **TypedElement** and thus can define the type of the value it can hold.
	
	Holder maps to a BPEL <variable>.
	BPEL uses three kinds of variable declarations: WSDL message type, XML Schema type (simple or complex), and XML Schema element.
	In the case of WSDL variable declaration, the <variable> element will be structured as follows:
	
	`<variable`
	`  name="[Process.Name]_Data"`
	`  messageType=`
	`  "[Process.Name]_ProcessDataMessage" />`
	
	The <message> element will be structured as follows:
	
	`<message name="[Process.Name]_ProcessDataMessage">`
	`  <part name="[Property.Name]"`
	`  type="xsd:[Property.Type]" />`
	`</message>`

Transaction	Open issue
Part Group	A <scope> provides the context which influences the execution behavior of its enclosed activities. This behavioral context includes variables, partner links, message exchanges, correlation sets, event handlers, fault handlers, a compensation handler, and a termination handler. Contexts provided by <scope> activities can be nested hierarchically, while the root context is provided by the <process> construct.
Comment from UML2 infrastructure	Can map to the <documentation> element. • If the **Comment** is associated with an object that has a straight-forward mapping to a BPEL element, then the text of the **Comment** will be placed in the <documentation> element of that object. • If there is no straight-forward mapping to any element, then the text will be appended to the <documentation> element of the <process>.
Simple Interaction.transformation	This will map to BPEL <assign> activities.

9.9 References

[BPEL11] ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf
[BPEL20] http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.pdf
[BPMN] http://www.omg.org/docs/dtc/06-02-01.pdf
[BPM-06-02] http://is.tm.tue.nl/staff/wvdaalst/BPMcenter/reports/2006/BPM-06-02.pdf
[RFC2119] http://www.ietf.org/rfc/rfc2119.txt

10 Proof of Concept Language Mappings

The following sub-sections describe mappings to specific languages as proofs of concept.

10.1 WS-CDL Mapping

[To be completed in a later version of this specification.]

Index

Abort Activity....54
Activity....5
Activity Loop....55
Actor....5, 56
Adhoc Process Directive....72
AdhocOrdering....73
Artifact Flow....73
Artifact Sequence Flow....73
Behavior....20
Behavior Event Condition....21
Behavior Model....11
Behavior Step....6, 21
Behavior Step Group....6, 22
Bindable Connection....22
Compensate Activity....75
Compensating Activity....75
Complex Decision....76
Complex Merge....76
Compliance....4
Compound Behavioral Connection....22
Compound Interaction....88
Compound Interaction Binding....89
Condition....7
Conditional Loop....56
Connected Part Binding....23
Course....7
Embedded Process....57
End Message....41
Error Activity....58
Event....7
Event Condition....7
Event Decision....77
Event Monitor....6, 23
Event Part....7
Exclusive Decision....77
Exclusive Merge....78
Gateway....7
Group Abort Connection....25
Holder....58
ImmediateSuccession....25
Inclusive Decision....79
Inclusive Merge....80
Instance: Abnormal End....26
Instance: Abnormal End Event....26
Instance: Abort....27
Instance: Abort Event....27
Instance: Abort Process....66
Instance: Activity Library....66
Instance: Activity Loop Behavior....66
Instance: Behavior Library....28
Instance: Behavior Occurrence....28
Instance: Cancel....83
Instance: Cancel Event....82
Instance: Cancel Process....83
Instance: Compensate....84

Instance: Compensate Event....84
Instance: Compensate Process....84
Instance: compensate-end....84
Instance: Compensation Library....85
Instance: Enclosed Step....30
Instance: end/abort....30
Instance: Error....31
Instance: Error Event....31
Instance: Error Process....66
Instance: Failure....33
Instance: Failure Event....32
Instance: Generalization....67, 85
Instance: Group Abort Behavior....33
Instance: group-step....34
Instance: ImportInfra....34
Instance: interationend-end....67
Instance: IterationEnd....67
Instance: IterationEnd Event....67
Instance: Normal End....35
Instance: Normal End Event....35
Instance: Process Occurrence....85
Instance: Racing Behavior....36
Instance: Racing Contestant....36
Instance: start-cancel....85
Instance: start-compensate....86
Instance: start-iterationend....68
Instance: start/start....36
Instance: StartFromSequence....86
Instance: startseq-end....86
Instance: Step Group....37
Instance: Success....37
Instance: Success Event....37
Interaction....6, 42
Interaction Protocol....7, 89
Interaction Role....6, 42
Interactive Behavior....43
Links Instance: cancel-end....83
LoopTestTime....59
Message....43
Message Flow....44
Multi Instance Loop....59
Performer Role....5, 60
Process....6, 61
Process Directive....80
Process Interaction Boundary....62
Processor Role....63
Race Connection....25
Received Intermediate Message....44
Role Realization....64
Script Activity....80
Sent Intermediate Message....45
Sequence Flow....80
Simple Activity....64
Simple Interaction....6, 46
Start Message....47
Sub-Process Activity....64
Substitutable Derivation....65
Succession....7, 25
Task....81

Terminate....81
Time Event....8
Time Event Condition....8