BUSINESS TRANSFORMATION AGENCY

Enterprise Architecture based on Design Primitives and Patterns

Guidelines for the Design and Development of Event-Trace Descriptions (DoDAF OV-6c) using BPMN

December 17, 2009

Version History

Version	Publication Date	Author	Description of Change
0.1	2009-01-26	Michael zur Muehlen	Initial Draft
0.2	2009-02-02	Michael zur Muehlen	Pre-Publication Draft
1.0	2009-02-03	Michael zur Muehlen	Initial Version for DoDAF Journal
1.1	2009-03-03	Michael zur Muehlen	Editorial Changes based on Review
1.2	2009-04-20	Michael zur Muehlen	Editorial Changes based on external Reader Feedback
1.3	2009-12-15	Michael zur Muehlen	Changes based on DoD DCIO Review
1.4	2009-12-17	Michael zur Muehlen	Changes Coordinated with DoD DCIO and DCMO

Table of Contents

Table of Figures

Acronym List

Acronym	Definition
BPDM	Business Process Definition Meta Model
BPM	Business Process Management
BPMN	Business Process Modeling Notation
BTA	Business Transformation Agency
DoD	Department of Defense
DoDAF	Department of Defense Architecture Framework
DM2	DoDAF Meta Model
PrOnto	Primitives Ontology
PriMo	Primitives Modeling Guide
OMG	Object Management Group
QUT	Queensland University of Technology
TU/e	Technical University of Eindhoven
UML	Unified Modeling Language
WfMC	Workflow Management Coalition
XML	eXtensible Markup Language
XPDL	XML Process Definition Language

Executive Summary

Enterprise Architecture (EA) is a key enabler of enterprise business process integration. While Architecture Frameworks such as DoDAF exist to guide the development of consistent architecture artifacts, significant roadblocks still exist for effective architecture development, adoption, integration, and federation.

Many of these roadblocks result from the lack of uniform representation for the same semantic content. Architects use different methodologies to develop models; these models are represented using different modeling languages and created using different modeling tools.

Even within a single methodology there may exist a variety of different modeling styles, techniques, and practices for similar content. Moreover, enterprise architecture is necessarily created by different organizations and disciplines. These in turn employ different terminologies that lead to different perceived business processes.

There is a need for standard formats for diagrams, standard data formats for the exchange of these diagrams, and standard formats for data that moves within and between the architectures that diagrams represent.

Our proposed solution is a set of architectural primitives and corresponding design patterns. These primitives and patterns provide a core set of 'building block' modeling elements founded in the well-defined semantics of the DoDAF Meta Model (DM2). These building blocks are accompanied by a recommended set of modeling techniques aimed at covering the different views on an Enterprise Architecture.

The Primitives/Lexicon Project has two core deliverables: A Core Ontology of Architectural Primitives (PrOnto) providing the basic vocabulary / lexicon of model elements and well-documented guidelines for modeling with Primitives (PriMo) delivering a comprehensive methodology for consistent model development.

This report describes guidelines for the development of Business Process Models using the Business Process Modeling Notation (BPMN 1.2). While other materials describe the syntax and semantics of the BPMN elements we focus on the relationship of these elements to the DM2 and their application to design models that are correct, consistent, and clear.

1 Introduction

1.1 Round-Trip Architecture

Enterprise Architectures are created using models that represent different aspects or views of an enterprise. A large number of competing techniques for the design of these models are in use at the DoD. As a consequence, there is currently no uniform representation for the same content in two architectures, unless the architects and modelers use the same technique, and apply this technique in the same way. The heterogeneity in tools and techniques is a necessary consequence of the different needs of architects that design organizational and technical systems. Due to backgrounds ranging from systems engineering to organizational change management, different modelers may perceive the same real world content differently. The problem does not lie exclusively with the modelers and architects. Tool vendors support a variety of techniques, and those that share techniques may support them only partially or with different visual representations.

The problem of inconsistent representation is not confined to the visual representation of architecture content. Architecture models are designed for human consumption, but ultimately they serve as input for the realization of the architecture. For this purpose the graphical models are converted into a machine-readable representation (e.g. an XML document) that can be read and executed by a suitable environment, such as a service orchestration platform. In order to support round-trip engineering this conversion has to work both ways. To date, round-trip engineering is mostly limited to vendor-specific solution stacks, using proprietary formats for model exchange. Standards for model exchange are the solution to overcome this limitation by allowing for interoperability between design and execution platforms of different vendors. A major issue in this area is the existence of multiple competing standards that bind the limited vendor resources to support such standards and fragment the market for interoperable tools.

1.2 Interoperability Issues

Even within a single standard solution problems persist. A major issue facing interoperability standards is the inconsistent implementation of existing specifications. Even though multiple tools may support the same diagramming technique, they may each use proprietary extensions. The same is true for the data formats used for model persistency. Figure 1-1shows an excerpt from an interoperability test scenario where Modeling tools used a common standard format to export and import process diagrams. Of the 38 feasible combinations only five worked without problems, while in eight scenarios the standard format was not understood at all by the receiving tool. In the remaining cases the original diagram was only partially read or significantly altered by the receiving tool, sometimes with a transformation of process semantics.

The consequence of this situation is the increasing difficulty to design integrated and federated architectures and ensure their use to build interoperable systems. It leads to communication gaps between model designers and decision makers, and leads to unclear relationships between the different views that are part of the DoD Architecture Framework (DoDAF).

1.3 Architectural Primitives

To solve this problem we propose the use of a rigorously defined set of core architectural primitives, rooted in the DoDAF V2.0 Meta Model (DM2), and the combination of these primitives into modeling building blocks (patterns) that can be reused across different projects and architectures. The primitives and patterns are complemented by a set of usage guidelines that help modelers create high-quality models.

The DM2 is a conceptual data model that represents the core data elements that should be described through architecture models, e.g. Capabilities, Activities, and Resources. Architecture models are created using modeling techniques that focus on particular aspects of the target system, e.g. UML Class Diagrams or Business Process Modeling Notation (BPMN) diagrams. These models are created using software tools for enterprise architecture or systems modeling.

Figure 1-1: Sample Interoperability Test

Compliance Test Matrix			Vendor	BizAgi	itPearls	Metastorm	Global 360	SunGard	Enhydra	Fujitsu
			Product	Process Modeler	Process Modeler for Visio	ProVision for Federal	Process Modeler Analyst Edition	Infinity Process Platform	JaWe	Interstage Busines Process Manager Studio
Vendor	Product	Prod. Vers.	Std. Vers.	0.66.1	5.0	6.1	1.0	4.5	2.4-1	8.1
BizAgi	Process Modeler	0.66.1	2.0	success	Gateway type changed to data-based from event-based	import failed	schema validation errors, no connectors	not supported	Gateways and intermediate events converted to activities	read, but no diagram
itPearls	Process Modeler for Visio	5	1.0	import failed	success	activities converted to gateways, no subprocesses	read, but no diagram	read, but no diagram	converted start/end event to activities	position of icons off, no subprocesses, sequence flow changed to condit. flow
			1.0 Carnot	import failed	only 2 activities	activities converted to gateways, no subprocesses	read, but no diagram	success	converted start/end event to activities	position of icons off, sequence flow changed to condit. flow
			2.0	2 activity symbols resized, position of icons mangled	success	positioning of icons inverted, no pools/lanes	import failed	not supported	converted start/end event to activities, no pools/lanes	icons inverted, scaling factor misaligned, sequence flow changed to condit. flow, no subprocesses
			2.0 Fujitsu	layout slightly off	success	import failed	import failed	not supported	Converted event to activity, added non-existent lanes, lost existing lanes	sequence flow changed to condit. flow, no lanes, sub-processes lost
Metastorm	ProVision for Federal	6.1	2.0	import failed	pool separator rendered as pool w/ objects, orig. pools are lost	Connector to Activity 3 rerouted	duplicate objects, misrouted connectors, no labels	not supported	import failed	scaling factor misaligned, sequence flow changed to conditional flow

The Primitives/Lexicon project bridges the gap between the content-agnostic modeling techniques and the requirements of the DM2 by mapping those constructs of the modeling techniques suitable to represent DoDAF architecture views to the concepts of the DM2.

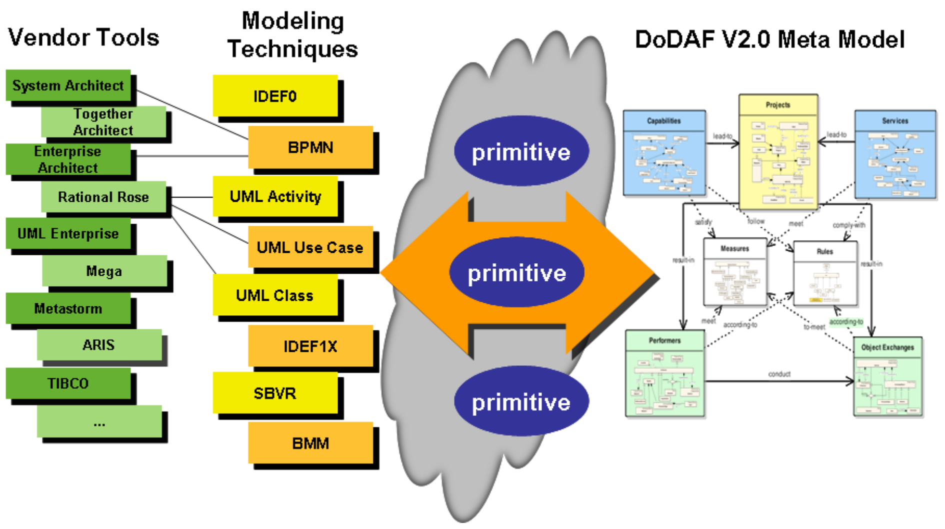
Figure 1-2 shows the relationship between the tools, techniques, and the DM2.

1.4 Desired Impact

The Primitives/Lexicon project strives to reduce the number of diagram types used in the construction of DoDAF-conformant architectures. We expect a number of results:

- Modelers have fewer modeling primitives to learn, as there will be a set of approved notations and notational elements to use.
- The limited number of modeling "dialects" will reduce the cognitive load both for model designers and model users. In other words, the resulting architecture views will be easier to understand for the trained user.

Figure 1-2: Positioning of Primitives



- A standardized representation of architecture modeling elements will enable the comparison of different architectures, which in turn enables the re-use of common modeling patterns and elements. This will lower the construction cost for enterprise architectures.
- A standardized set of modeling methods enables the standardized training of model designers and users which will make it easier to bring team members into new projects, and will expand the potential qualified labor pool for DoD projects.

Ultimately, a standard set of architecture views and modeling techniques supported by structured training efforts will lead to a higher quality of architecture products, which in turn improves the overall quality of the architecture in general, which ultimately will lead to a higher solution quality.

1.5 No New Notation

It is important to point out that the Primitives/Lexicon project does not develop new modeling methods or notations. Instead, it matches existing notations to DoDAF views and DM2 data elements, and develops guidelines for the creation of high-quality models using these notations.

These guidelines will vary based on the maturity of the respective DoDAF view, notation, and established modeling practices. The OV-6c/BPMN combination is the first area for these guidelines, others will follow.

2 Quality Criteria for Architecture Models

The goal of this modeling guide is to facilitate the design of correct, high-quality architecture models. In order to achieve this goal we use a model quality framework to illustrate the quality requirements for architecture models.

The quality of architecture models has a direct impact on how architecture can be communicated and implemented. High-quality models are easier to read, faster to understand, and thus will cause fewer errors at the implementation level. Lower quality models take more time to comprehend and may contain misleading or ambiguous elements that may result in implementations that do not meet the functional and non-functional requirements of the client.

We use a framework that defines the criteria for high-quality models based on the published Guidelines of Modeling. These criteria are Correctness, Relevance, Cost-Effectiveness, Clarity, Comparability, and Systematic Design.[1]

2.1 Correctness

At a minimum, architects must ensure that in the design of their architectures they create correct models. This correctness is defined by the following properties:

- *Syntactical Correctness*, i.e. the model satisfies the rules of the modeling language: A model must satisfy the vocabulary and grammar restrictions of the modeling language chosen. In particular this correctness means two things:
    - The model uses only approved modeling constructs: A model must not contain any constructs that are not part of the chosen modeling language. Some languages, such as UML, allow for the customization of model types through stereotyping. In these cases it is important that the customizations used are documented and agreed upon by the architect community.
    - The modeler connects these modeling constructs in permissible ways, i.e. only in those ways that are permitted by the modeling language. For example, a process modeler using BPMN must not use message flow arrows to connect activities that reside within the same pool.
- *Semantic Correctness*: The model satisfies the semantic requirements of the problem domain depicted
    - Factual correctness: The model captures the problem domain accurately, i.e. it does not misconstrue reality.
    - Appropriate level of detail: The model captures aspects of the problem domain at a sufficiently detailed level to be actionable for the model user, and at a sufficiently abstract level to reduce the complexity of reality to a manageable level for the model user.

However, just satisfying the correctness requirement does not necessarily result in a good model. A good architecture model satisfies the correctness requirements and the following additional criteria:

2.2 Relevance

- *Problem Relevance*. An architecture model has to contain content that is relevant to the problem domain surveyed. Note that problem relevance is a subject-specific, or more precisely, recipient-specific criterion. This criterion can be described in two dimensions: Coverage and size. While coverage should be maximized, the size of the model should be minimized.

---

[1] See for example:
Becker, J., Rosemann, M. & von Uthmann, C.: Guidelines of Business Process Modeling. In: Business Process Management. Models, Techniques, and Empirical Studies, (Eds, van der Aalst, W.M.P., Desel, J. & Oberweis, A.) Springer, Berlin, Germany, 2000, pp. 30-49.
Moody, D.L.; Shanks, S.: What Makes a Good Data Model? Evaluating the Quality of Entity Relationship Models. In: Loucopoulos, P. (Eds.): Entity-Relationship Approach - ER'94. Business Modelling and Re-Engineering. 13th International Conference on the Entity-Relationship Approach. Berlin, Heidelberg etc.: Springer 1994, pp. 94-111.

    - *Maximum coverage* means that an architecture model has to capture all relevant aspects of the problem domain. It should not omit any content that a model user may require to fully understand the problem
    - *Minimum size* means that a model should not contain details that are irrelevant to the model users.

2.3 Cost-Effectiveness

- *Cost-effective Design*. The creation of architecture models should not incur prohibitive construction cost. The cost of analyzing the underlying problem space and the creation of the models should not outweigh the benefits derived from the use of the models. The use of reference models as templates, the assembly of predesigned modeling patterns or fragments, and the re-use of models or model elements can contribute to a lowering of construction cost, making the design of complex models economically feasible.

2.4 Clarity, Comparability, and Systematic Design

- *Clear, Comparable, Systematic Design*. Architecture models that are difficult to understand may not be used – the cognitive effort required by the reader may outweigh any potential benefits of using the model. The three criteria of clarity, comparability and systematic design evaluate to what extent an architecture model has been designed with simplicity and readability in mind. A model that satisfies these criteria will be easier to understand and to use than a model that violates either one of the three criteria.
    - *Systematic Design* means that the results of the architecture design process should follow a systematic layout. In the case of process models this means that the flow direction should be uniform across all models (left-right or top-down).
    - *Clarity of design* means that crossing lines and overlapping symbols should be avoided wherever possible.
    - *Comparable design* means that related content should be arranged in a similar fashion so as to enable a cross-check to uncover structural analogies. For example, a data model for suppliers (accounts payable) should be structured in a similar fashion to the data model for customers (accounts receivable).

3 Modeling Primitives

3.1 What is a Modeling Primitive?

Architectural primitives describe elementary language building blocks for architecture products. These primitives are directly related to core elements of the DoDAF Meta Model. Primitives bridge the gap between the core DM2 architecture concepts and their associated architecture models, between the architectural models and the various methods/techniques for modeling them, and between the modeling methods/techniques and standard presentations for the architecture concepts.

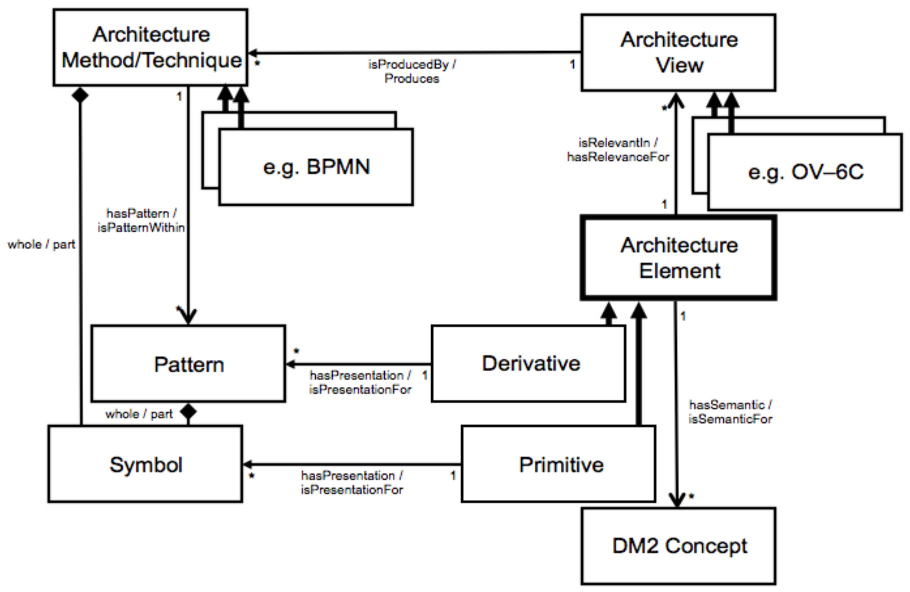
Ultimately a primitive identifies a standard presentation for rendering a core DM2 concept within a particular method/technique within an architecture model. A primitive could be rendered differently in different architectural methods/techniques, but always refers to the same DM2 concept. For example, in a DM2 *performer* would be rendered as a stick figure in a UML use case diagram, and as a swimlane in a BPMN diagram.

Primitives allow for the transformation of models that use different modeling techniques.

3.2 Ontology Representation

An ontology is a formal representation of a set of concepts and the relationships between those concepts within a domain.[2]Figure 3-1 illustrates the basic concepts and relationships used to define the Lexicon/Primitives domain. The central concept in this ontology is the Modeling Element, which represents a basic building block for developing enterprise architecture. We describe both primitive modeling elements (as described above) with corresponding atomic representation symbols, as well as and associated derivative modeling elements that correspond to low-level design patterns built on the basic primitive elements. Modeling Elements are related to core architecture concepts from the DoDAF V2.0 Meta Model in order to define the semantics of a modeling element 'building block'.

Figure 3-1: Lexicon/Primitives Ontology

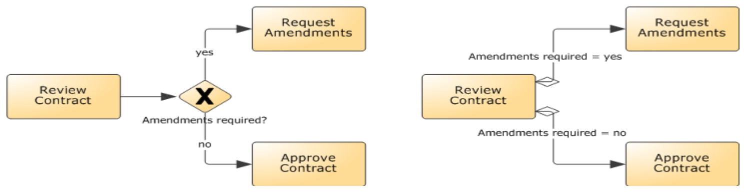

Architecture Method/Technique
isProducedBy / Produces
Architecture View
e.g. BPMN
isRelevantIn / hasRelevanceFor
e.g. OV–6C
hasPattern / isPatternWithin
whole / part
Architecture Element
Pattern
hasPresentation / isPresentationFor
Derivative
whole / part
hasSemantic / isSemanticFor
Symbol
hasPresentation / isPresentationFor
Primitive
DM2 Concept

[2] See e.g. Guarino, N.; Oberle, D.; Staab, S.: What is an Ontology? In: S. Staab & R. Studer. Handbook on Ontologies. 2nd revised edition. Springer, 2009.

4 Using BPMN to Model OV-6C Event-Trace Descriptions

Figure 4-1 shows an example of two alternative representations of the same process. Both diagrams express the same semantics: After a contract review either contract amendments are requested or the contract is approved.

Figure 4-1: Alternative BPMN Solutions Example



Both renditions are valid models in the Business Process Modeling Notation (BPMN) 1.2. But the model on the left uses a gateway (diamond) symbol that indicates that the following activities are mutually exclusive, whereas the model on the right uses conditional flow elements to express this semantic. In order to understand the relationship between the two conditional flow connectors the user has to parse the associated conditions and infer whether they are mutually exclusive or potentially overlapping. Without any guidance modelers are free to choose either representation, leading to models that mix both representations. This makes the models harder to understand by users, which in turn can lead to inconsistent implementations of architecture products due to misinterpretations of the depicted semantics.

In order to reduce the variability of visual representations of the same content and consequently the ambiguity of the resulting models two steps are required:

First, a reference set of elementary modeling elements needs to be defined, eliminating some of the duplicate ways of representing a given scenario. For example, by eliminating the conditional flow element from the list of allowable modeling elements we can eliminate the representation on the right of figure 4-1. However, the elimination of modeling constructs must not compromise the expressiveness of the language as it is required for a particular purpose.

For instance, BPMN contains the notion of a compensation activity that is useful when specifying transactional behavior at the system level. However, the OV-6c targets the conceptual or requirements specification level, thus the compensation activity can be eliminated for the OV-6c without negatively affecting the applicability of BPMN. At the level of the SV-10c (a systems event-trace description) handling compensation behavior may be essential, thus the BPMN subset for an SV-10c model may contain more symbols than the BPMN subset for an OV-6c model.

Once a set of reference modeling elements has been determined, the use of these elements needs to be standardized. We propose the use of modeling patters both at an elementary level (to ensure model correctness), and at a semantic level (to ensure standard solutions to common problems).

In the following sections we will address both the selection of a BPMN subset for the OV-6c as well as the design of patterns for use by BPMN modelers.

4.1 BPMN Development Methodology

Federated Architectures rely on the consistent use of Architecture Frameworks and the systematic design of architecture views. A standardized development process is thus necessary to facilitate the a process for the development of a core subset of architecture products.

BPMN lends itself to a top-down approach to process analysis. The BPMN standard contains provisions that allow for the transition of BPMN models into executable environments, but this transition works best if the models have been designed in a consistent fashion. At the BTA, for example, most BPMN models are designed for communication purposes. In order to make a model suitable for communication a few guidelines should be followed:

Model size: The BPMN model should be limited in size, both in terms of physical size and in terms of the number of elements contained in the model. Wallpaper-size process diagrams are typically an indicator of unclear horizontal and vertical process separation.

Vertical separation means that a model should fit within one well-defined level of abstraction. This implies that all activities contained in the model are of similar granularity and abstraction. A useful indicator to establish a consistent level of abstraction is the use of process objects, i.e. the key data elements, documents, or files that flow through a process and are manipulated in the individual activities. If one step operates on a set of documents while the next step operates on an attribute of a data element, these steps are not at the same level of abstraction.

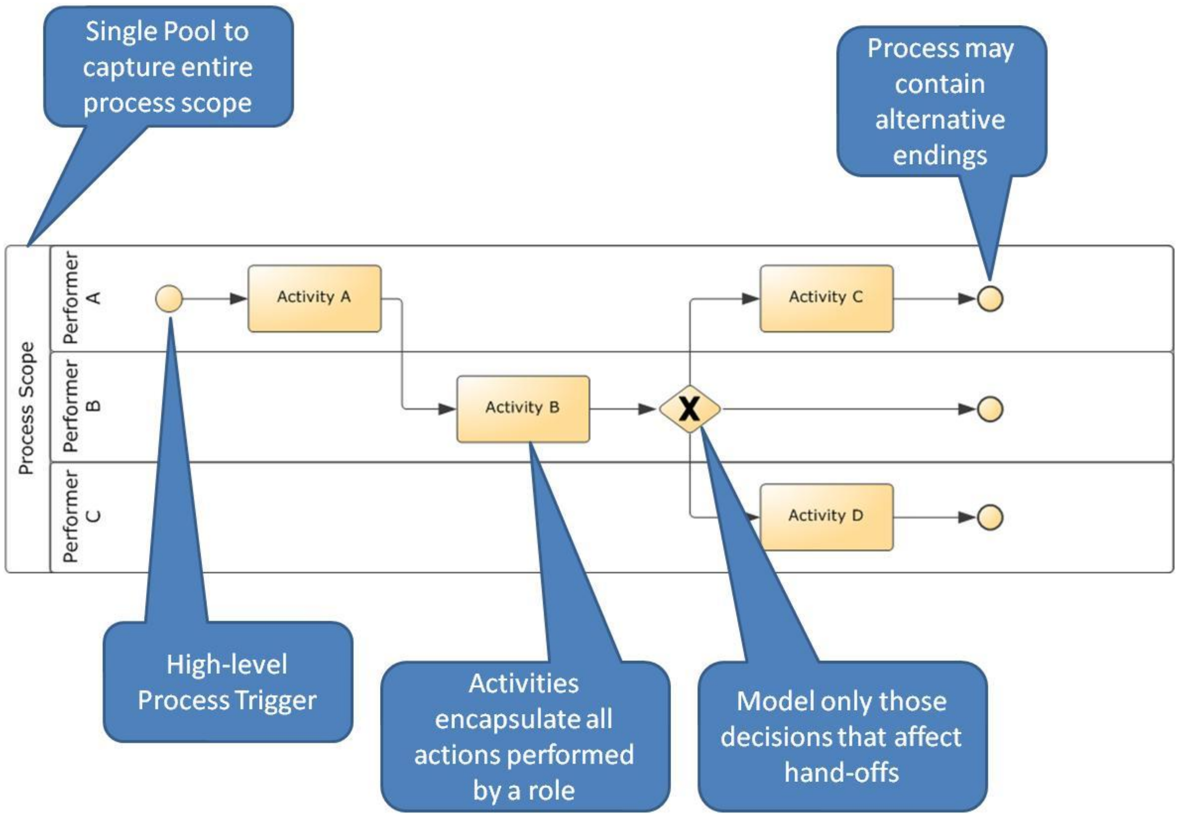
Horizontal separation means that an end-to-end process should be broken into multiple concatenated diagrams, unless it is described at the highest level of abstraction (typically as a value chain). Again, process objects can be helpful in determining handoff points between process segments.

We recommend three levels of abstraction for OV-6c models: Handoffs, milestones, and procedures.

4.1.1 Handoff-Level Processes

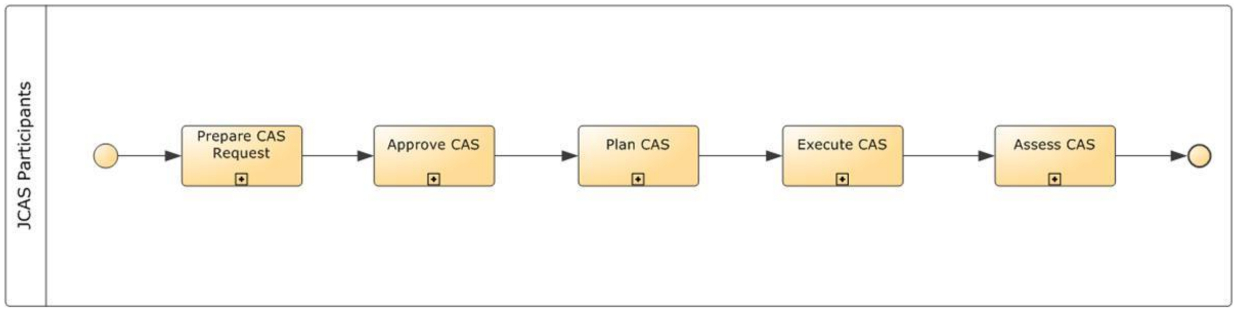
At the handoff level the process is broken into activities that group all actions by a performer until a handoff to another performer occurs. The entire process should be contained in a single pool that demarcates the process scope. The main focus of the handoff level is to establish the process boundaries, the roles involved in the performance of the process, and the major communication points (i.e., handoffs). The handoff level process diagram can contain multiple process outcomes (e.g., success, failure).

Figure 4-2: Handoff-Level BPMN Process


Single Pool to capture entire process scope
Process may contain alternative endings
Process Scope
Performer A
Performer B
Performer C
Activity A
Activity B
Activity C
Activity D
X
High-level Process Trigger
Activities encapsulate all actions performed by a role
Model only those decisions that affect hand-offs

In some cases, e.g. in highly collaborative processes, it may not be possible to define clear handoff-points because of the amount of back-and-forth communication between parties. In this case it is advisable to focus on the major process milestones and model these within a pool without identifying individual performers. For example, the process diagram below shows the major milestones of a Joint Close Air Support mission thread, independent of the performers that carry out these steps.

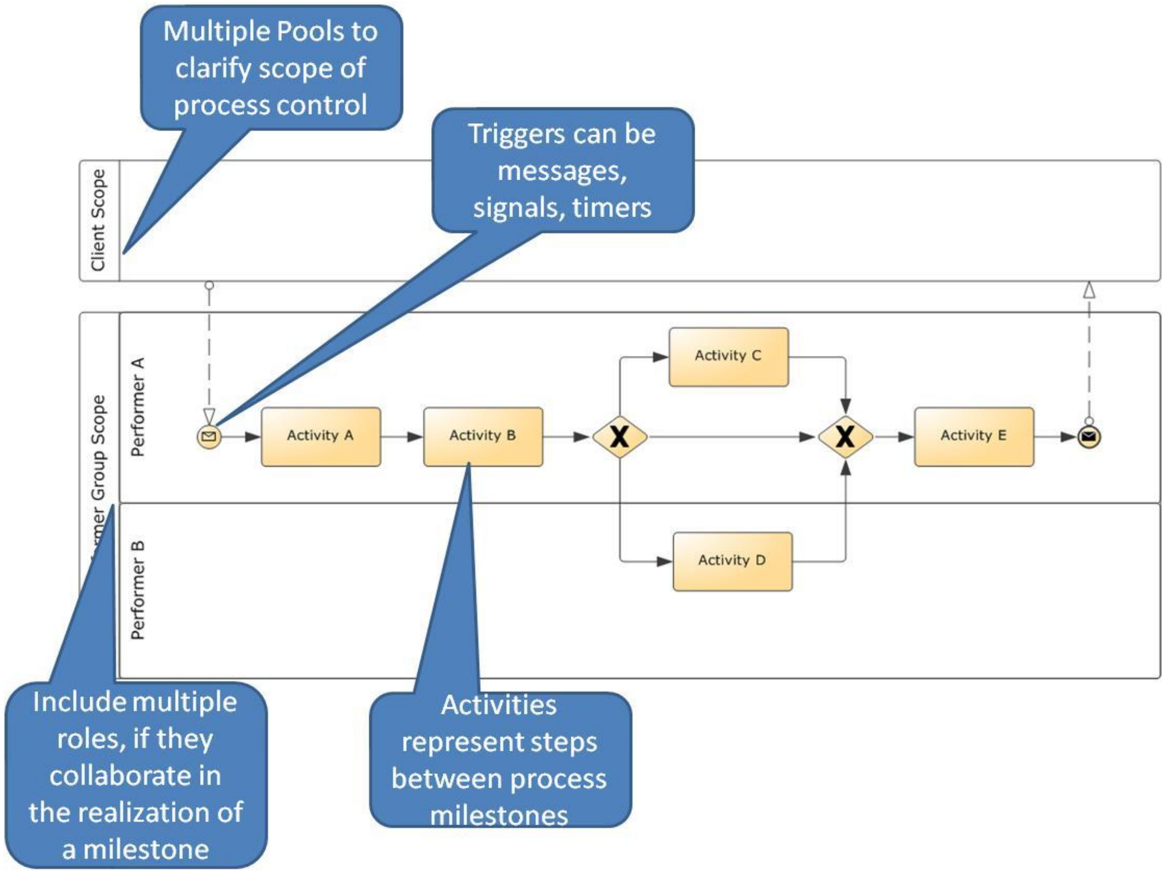
Figure 4-3: Example Top-Level BPMN Process



4.1.2 Milestone-Level Processes

At the milestone level, each activity of the handoff level is broken into activities and decisions that affect the flow of the process in a significant way. The milestone level process diagram will typically contain multiple swimlanes for different performers; however, one of these swimlanes will likely be the dominant performer due to the nature of the decomposition. If external parties are involved in the process they can be represented using additional pools with appropriate message-based linkages to the main pool.

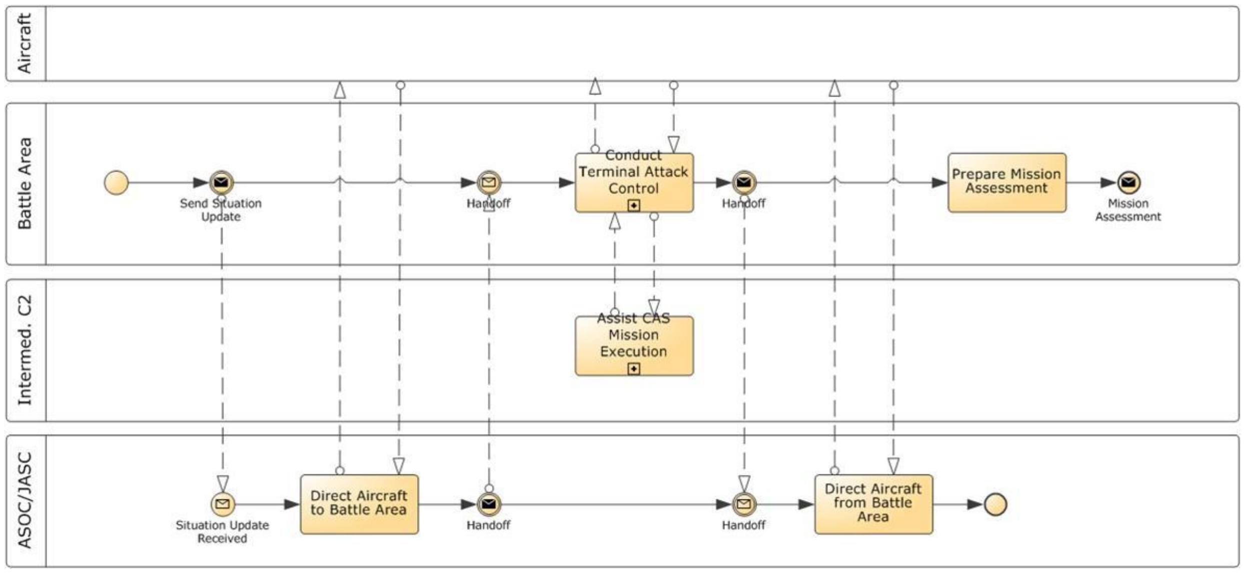
Figure 4-4: Milestone-Level BPMN Process


Multiple Pools to clarify scope of process control
Triggers can be messages, signals, timers
Client Scope
Performer Group Scope
Performer A
Performer B
Activity A
Activity B
Activity C
Activity D
Activity E
Include multiple roles, if they collaborate in the realization of a milestone
Activities represent steps between process milestones

The focus of the milestone-level process is on measurable state changes in the overall process. Typically, the achievement of a milestone is a measurable event that can be used for reporting purposes. For this reason, milestone-level processes may focus on the messages that a process participant sends or receives.

For instance, the process diagram below shows the execute phase of a Joint Close Air Support mission thread, with an emphasis on the messages exchanged by the individual participants.

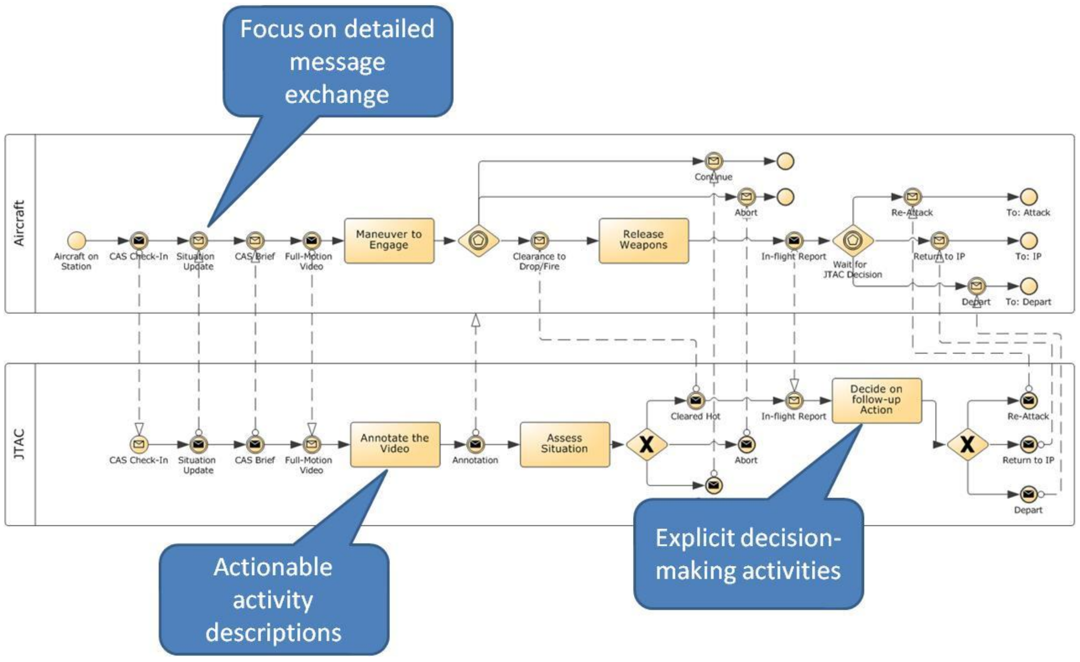
Figure 4-5: Example Milestone-Level BPMN Process


Aircraft
Battle Area
Intermed. C2
ASOC/JASC
Send Situation Update
Handoff
Conduct Terminal Attack Control
Handoff
Prepare Mission Assessment
Mission Assessment
Assist CAS Mission Execution
Situation Update Received
Direct Aircraft to Battle Area
Handoff
Handoff
Direct Aircraft from Battle Area

4.1.3 Procedure-Level Processes

While the handoff level and the milestone level both focus on the process content in terms of *what* needs to be accomplished, the procedure level focuses on *how* individual activities are carried out. It represents the closest bridge to the SV-10c process diagrams in that it can contain explicit references to messages, data formats, and technical systems used to carry out individual activities.

Figure 4-6: Example of Procedure-Level BPMN Process



The example below shows a detailed diagram of the JCAS mission thread activity "Conduct Terminal Attack Control" that was depicted as a single activity in the milestone-level BPMN process. At the procedure level the diagram outlines the messages that different participants are exchanging, explicitly outlines decision making activities, and describes operational activities at an actionable level.

4.2 BPMN Symbol Subset for OV-6c Event Trace Descriptions

The atomic primitives and associated low-level design patterns describe a set of *normative* modeling elements that restrict the use of BPMN. This means that modelers cannot use alternative representations for the concepts described in this section.

BPMN 1.2 defines 53 modeling symbols. Not all of these symbols are similarly important. While nearly all BPMN models contain tasks and sequence flow, very few models use constructs such as compensation, transaction boundary, and intermediate rule events. For the purposes of the BTA we analyzed the occurrence of symbols in the Business Enterprise Architecture (BEA) and developed BPMN models of the Joint Close Air Support (JCAS) use case within the Core Enterprise Services to the Tactical Edge (CES2TE) project, as well as a reference process for the Global Collaborative Manufacturing Architecture (GCMA) project. Based on our experiences in these three areas we developed a recommended set of BPMN constructs for use as the OV-6c building blocks.

This core set of BPMN constructs reduces the vocabulary of BPMN 1.2 from 53 to 29 elements. The omitted symbols fall into three groups:

- Elements that relate to execution-level semantics (e.g. transactions, compensation) were omitted, because the views created from the OV-6c model should be at an implementation-agnostic level. Execution semantics play a role at the systems level (e.g. SV-10c) or services level (e.g. SvcV-10c), but should be avoided at the OV-6c level.
- Diagram embellishments without semantics of their own (e.g. group, annotation) are often abused by modelers to capture process semantics that are difficult to model using other constructs. Since these embellishments do not translate into executable models (e.g. when BPMN models are translated to BPEL), the semantics contained in these annotations will be lost, potentially resulting in faulty models. Annotations and embellishments can be created through functionality provided by the modeling tools, but do not constitute core BPMN primitives.
- Elements that can be represented by other constructs without loss of semantics (e.g. complex gateway). For these elements we chose one standard representation. For instance, the BPMN standard provides conditional sequence flows as an alternative to the use of gateways. But conditional sequence flow is fraught with problems at the diagram level: If an activity has several outgoing conditional sequence flows it is impossible for the reader to determine whether these flows are mutually exclusive or overlapping (XOR or OR semantics) without looking at the attributes of the conditional sequence flow elements – and these attributes are not rendered at the graphical level. This can lead to misinterpretations of the model, and complicates the design of joins (merges) that mirror the semantics of the preceding splits.

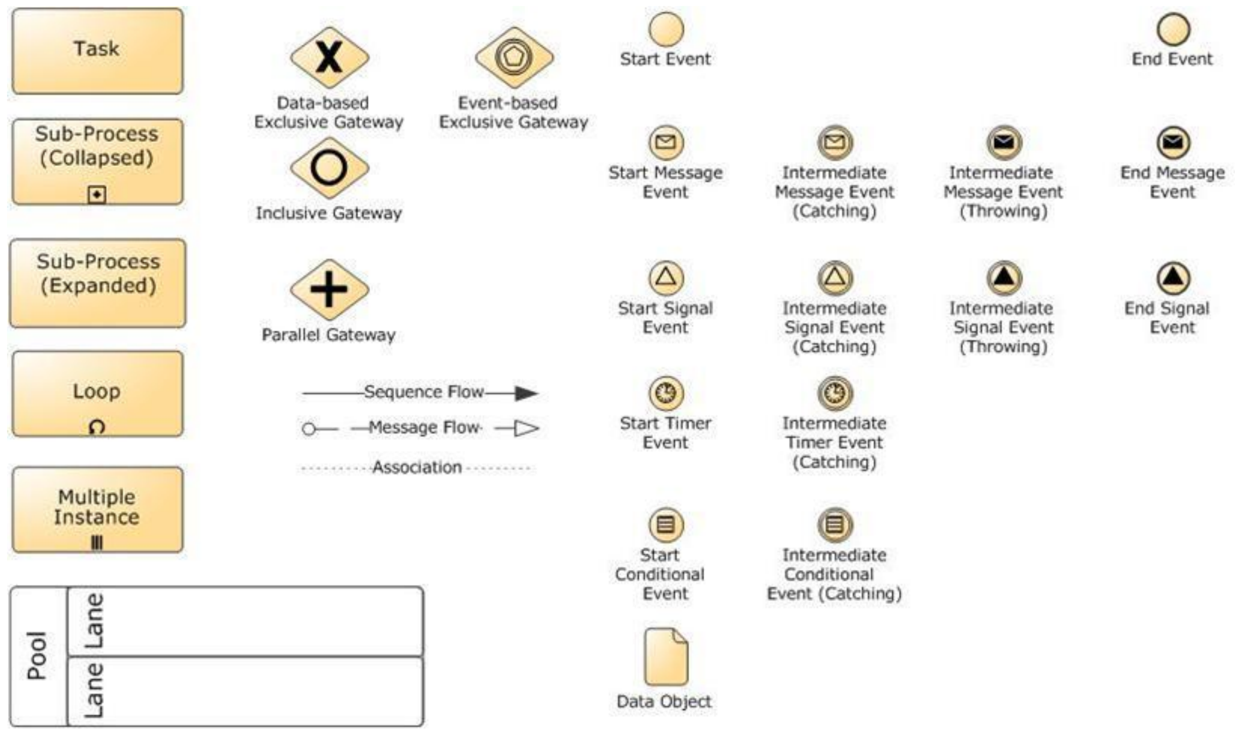
A detailed description of these primitives, their semantics, and their linkage to the DM2 is provided in Appendix A.

The most significant elimination took place in the area of events. For the purposes of the BTA we limit the use of BPMN to message, timer, conditional, and signal events. Message events are used to represent point-to-point communication. Signal events represent broadcast communication. Both are essential to depict radio traffic and other kinds of information exchange. Timer events are essential to represent scheduled activities as well as planned delays and timeouts. Conditional events are useful to capture rule-based behavior in the process model and provide a link to the OV-6a Operational Rules Model.

Figure 4-7 shows the BPMN primitives that resulted from our analysis of BPMN usage in theory and practice.

Note that not all process modeling toolsets support these constructs. Those tools that are BPMN 1.0 compliant do not provide signal events, nor do they distinguish between throwing and catching intermediate messages. Since the BPMN 1.2 specification was published in January 2009, these missing elements should be available through software updates in the near future. In the meantime modelers can use intermediate events and message events to approximate the semantics of the missing elements while preserving an upgrade path, once their tool reaches BPMN 1.2 compliance. The BPMN 2.0 specification – currently within the Object Management Group Finalization Task Force – will introduce new symbols for the modeling of choreographies and new event types, and the primitives specification will be updated once this version of the standard has been officially released by the Object Management Group.

Figure 4-7: BPMN Symbol Subset (Primitives)



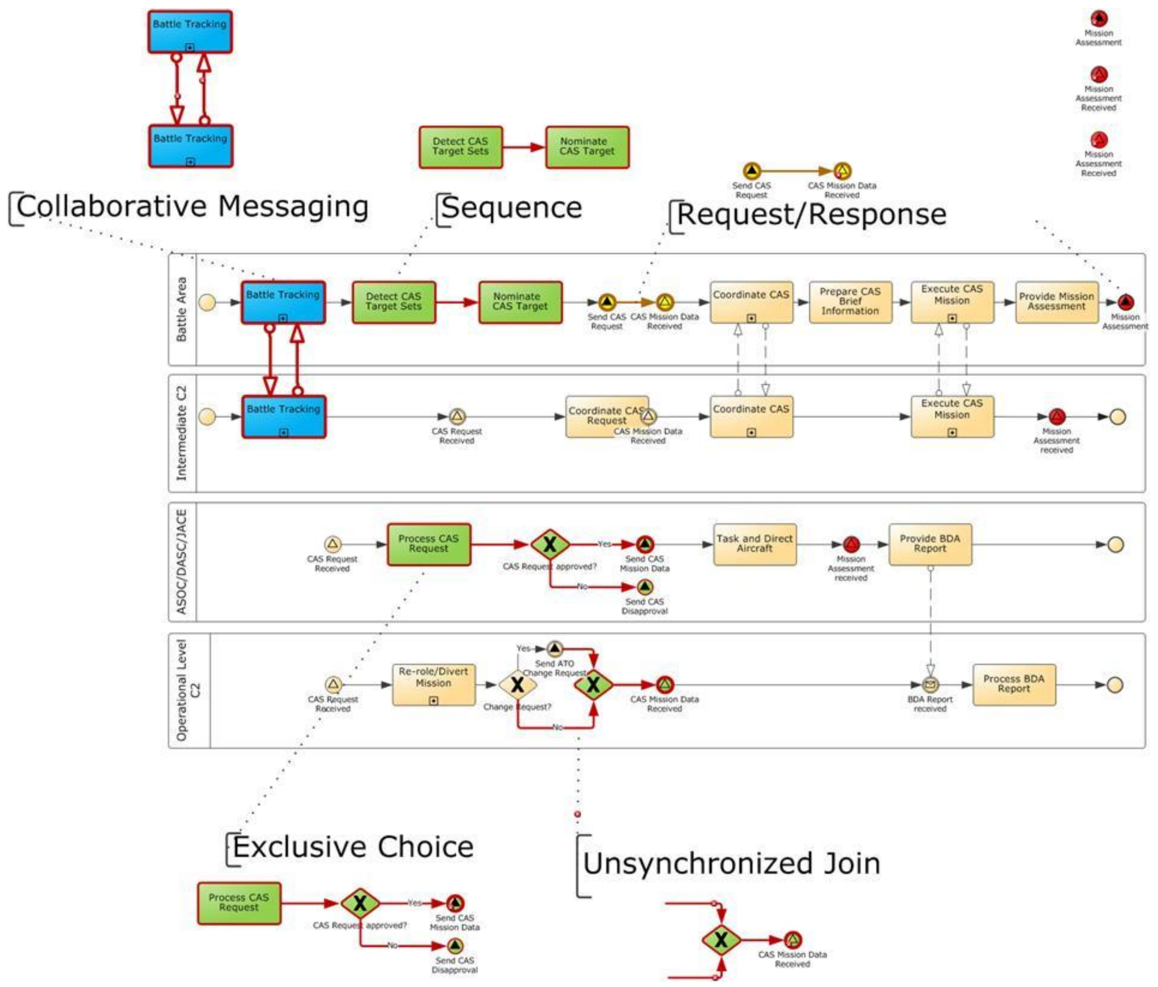
4.3 BPMN Design Patterns

As discussed earlier, limiting the BPMN 1.2 vocabulary provides only a partial solution to the problem of inconsistent models. In order to facilitate consistent, correct, and clear models, design patterns provide guidance for modelers that need to capture typical process semantics. Figure 4-8 shows such design patterns in the context of a larger JCAS mission thread BPMN diagram. Note how the process is composed of recurring patterns, while at the same time the process is constructed using the BPMN 1.2 subset described in the previous section.

The BPMN design patterns support the construction of high-quality models. For this purpose we have developed two sets of design patterns. Low-level design patterns provide elementary process flow semantics, such as sequence, split and join. These low-level patterns represent an exhaustive list of design options based on several research projects and will remain invariant. Their use is mandatory for architects that want to design primitives-compliant processes. High-level design patterns provide solutions to recurring design problems that encapsulate semantics such as multiple start events, state-like activities, process synchronization and so on. By nature, these design patterns are not exhaustive, and we expect this catalog of patterns to grow over time as process designers begin to contribute their own solutions based on individual project experience. The primitives design team welcomes such submissions and we will publish them in subsequent DoDAF Journal entries.

The following two sections describe in detail the BPMN design patterns, both low-level and high-level. In addition, Appendix B contains a table of the low-level design patterns and their mapping to DM2 elements.

Figure 4-8: Design Patterns in Context



5 Low-Level BPMN Design Patterns

The following low-level control flow patterns are based on the Workflow Control Flow Patterns work by the Technical University of Eindhoven and Queensland University of Technology.[3] They represent elementary process building blocks and are chosen to provide a unified representation for common process model fragments. For each pattern we provide the semantics of the pattern, the rationale (i.e. why this pattern is needed) and design guidelines that cover the use of the pattern.

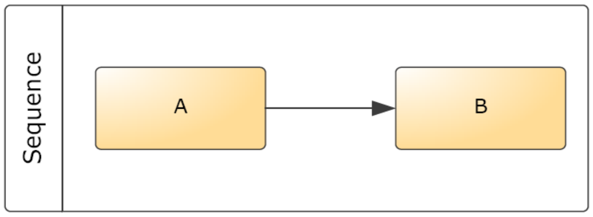
5.1 Elementary Patterns

Elementary patterns are the foundational building blocks of a BPMN process diagram. In this section we list each pattern, outline its semantics in plain English, provide a rationale for why the pattern is needed, and specify design guidelines for the construction of the pattern.

It should be noted that some of the design guidelines are more restrictive than the BPMN specification. These restrictions are given to facilitate clear and unambiguous representations of processes and process patterns.

5.1.1 Sequence [WCP-1]

Figure 5-1: Sequence Pattern



5.1.1.1 Semantics

Activity A needs to complete before Activity B can start. A sequence flow indicates a dependency between two activities, in the above case Activity B depends on Activity A in some form – either through a shared performer or the exchange of data. A sequence flow always implies a data flow; however, the inverse is not necessarily true – an activity may produce data that may be supplemental for another activity, but not required.

5.1.1.2 Rationale

BPMN diagrams are intended to capture flow dependencies between activities. The sequence pattern represents the most basic dependency of two activities.

5.1.1.3 Design Guide

- The general modeling direction for BPMN diagrams should be horizontal left to right, thus the sequence flow should follow this general direction.[4] Exceptions are permissible when sequence flow is used to loop back to a previously executed activity.

[3] N. Russell, A.H.M. ter Hofstede, W.M.P. van der Aalst, and N. Mulyar. Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22, BPMcenter.org, 2006. Note that this revised report identifies more patterns and thus supersedes the original Workflow Patterns work published as W.M.P van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. Distributed and Parallel Databases, 14(3), pages 5-51, July 2003

[4] In cases where a vertical modeling orientation is necessary (e.g. due to space constraints) similar consistency should be enforced. A mix between horizontal and vertical diagram orientations within the same context should be avoided.

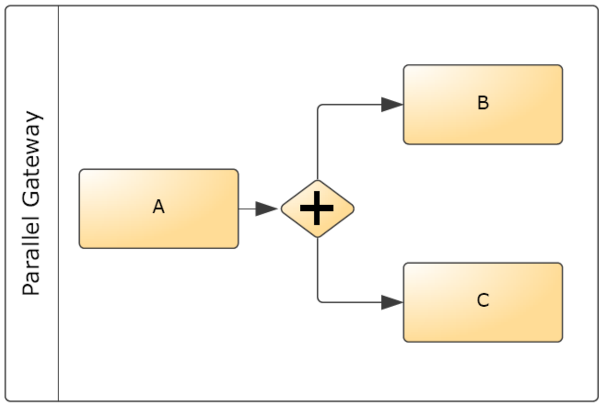
- The outgoing Sequence Flow connects on the right side of the preceding Activity
- The incoming Sequence Flow connects on the left side of the succeeding Activity.
- Sequence Flow connections from/to the top and bottom of activities should be avoided.

5.2 Split Patterns

Split patterns are used to create multiple branches of control flow in a process. Depending on the semantic of the split these branches can either represent independent pathways (i.e. concurrent threads) or they represent alternative processing pathways.

5.2.1 Parallel Split (AND-Split) [WCP-2]

Figure 5-2: Parallel Split Pattern



5.2.1.1 Semantics

After completion of Activity A both Activity B and Activity C can start independent of each other.

Note: The AND split does not imply that B and C have to occur at the same time. If the same performer is responsible for B and C these activities might be performed sequentially, although the sequence (B before C or C before B) is not constrained.

5.2.1.2 Rationale

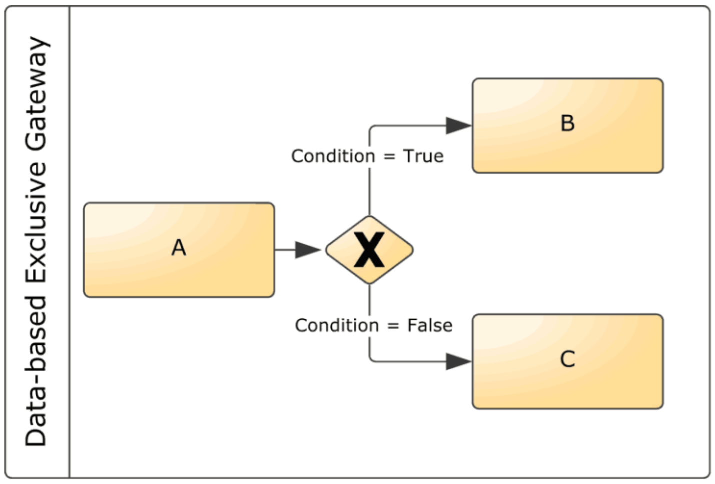
One of the most prominent sources for efficiency improvements is the parallelization of independent tasks. This pattern represents the beginning of such a parallelization. It is frequently combined with the synchronized AND join pattern [WCP-3] for the merging of such parallel flows.

5.2.1.3 Design Guide

- The incoming Sequence Flow connects on the left side of the AND Split Gateway.
- All outgoing Sequence Flows from the AND Split Gateway originate right of the middle of the symbol.
- The AND Split Gateway displays the + symbol to distinguish it from other types of gateways.
- The AND Split Gateway can have an arbitrary number of outgoing sequence flows, but must have at least two outgoing sequence flows.
- The AND Split Gateway can have only one incoming sequence flow.

5.2.2 Exclusive Choice (XOR-Split) [WCP-4]

Figure 5-3: Exclusive Choice Pattern



5.2.2.1 Semantics

After completion of Activity A either Activity B or Activity C can start (but not both), depending on the truth-values of the Condition.

Only one outgoing sequence flow from the XOR Gateway can evaluate to true.

5.2.2.2 Rationale

BPMN is capable of capturing multiple process execution scenarios in a single diagram (as opposed to UML Sequence Diagrams which are limited to one scenario per diagram). The exclusive choice pattern is needed in cases where the execution of process activities varies depending on the run-time scenario. This pattern is often combined with the unsynchronized XOR join pattern [WCP-5 or 8], which merges the control flow from two or more alternative threads.

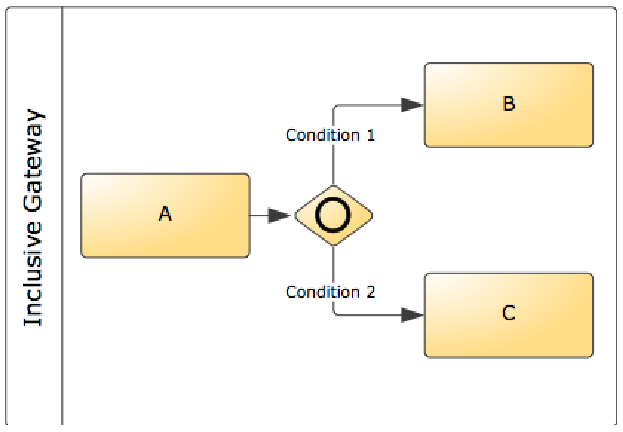
5.2.2.3 Design Guide

- Within readability constraints, label the outgoing sequence flow with condition descriptions
- The XOR Choice Gateway displays the X symbol to distinguish it from other types of gateways.[5]
- The XOR Choice Gateway can have an arbitrary number of outgoing sequence flows, but must have at least two outgoing sequence flows.
- The XOR Choice Gateway can have only one incoming sequence flow.
- The XOR Choice Gateway only interprets the results of a previous activity; it does not represent a decision-making activity. If a manual or automated decision-making step is required to set the value of the condition, this step must be modeled as a separate activity that precedes the XOR Choice Gateway. In the example above Activity A is a decision-making activity, whereas the XOR Choice Gateway just evaluates the outcome of the decision-making activity.
- The default case (i.e. the standard outcome) can be highlighted by using the Default Sequence Flow symbol.

[5] While BPMN permits the use of blank gateway symbols we recommend the use of explicit gateway markers to improve the readability of BPMN diagrams.

5.2.3 Multiple Choice (OR-Split) [WCP-6]

Figure 5-4: Inclusive Choice Pattern



5.2.3.1 Semantics

After completion of Activity A either Activity B or Activity C or both Activity B and C can start, depending on the truth values of Conditions 1 and 2.

Multiple outgoing sequence flows from the OR Gateway can evaluate to true, i.e. it is possible to depict n out of m semantics.

5.2.3.2 Rationale

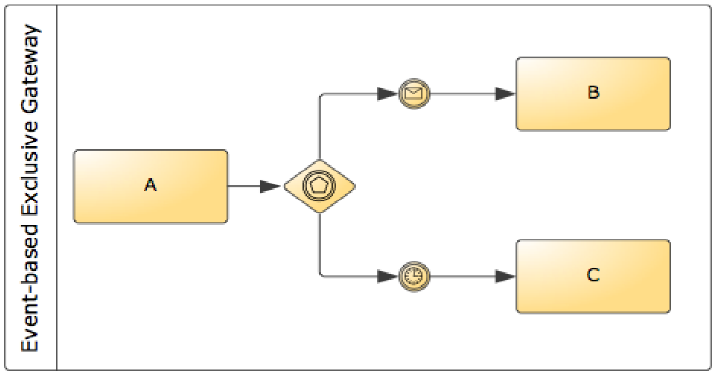
BPMN is capable of capturing multiple process execution scenarios in a single diagram. The inclusive choice pattern is needed in cases where the execution of process activities varies depending on the scenario, and where one or more branches of a process may be executed in a given scenario. This pattern is often combined with the synchronized OR-join pattern [WCP-7 or 9], which merges the control flow from two or more threads that have been split with the inclusive choice pattern.

5.2.3.3 Design Guide

- Within readability constraints, label the outgoing sequence flow with condition descriptions
- The Inclusive Choice Gateway displays the O symbol to distinguish it from other types of gateways.
- The Inclusive Choice Gateway can have an arbitrary number of outgoing sequence flows, but must have at least two outgoing sequence flows.
- The Inclusive Choice Gateway can have only one incoming sequence flow.
- The Inclusive Choice Gateway only interprets the results of a previous activity; it does not represent a decision-making activity. If a manual or automated decision-making step is required to set the value of the condition, this step must be modeled as a separate activity that precedes the Inclusive Choice Gateway. In the example above Activity A is a decision-making activity, whereas the Inclusive Choice Gateway just evaluates the outcome of the decision-making activity.

5.2.4 Event-based Choice (Event-based XOR-Split) [WCP-16]

Figure 5-5: Event-based Choice Pattern



5.2.4.1 Semantics

After completion of Activity A the Process will halt until one of the subsequent events occurs. In the sample diagram above, the process will halt until either a message is received or a timer expires. If the message is received first, Activity B can start and the timer is disabled. If the timer expires first, Activity C can start and the message receiver is disabled.

Only one outgoing sequence flow from the Event-based Gateway can evaluate to true at any given time, thus the Event-based Gateway behaves like an XOR-Gateway, and the separate process paths can later be merged with an XOR-Join (unsynchronized join, see below).

5.2.4.2 Rationale

A process may have to react to changes in its environment. Intermediate catching events are listeners that sense environmental changes (arrival of messages, expiration of timers, presence of signals etc.). If a process will react to just one out of a set of environmental changes the Event-based Gateway reflects this situation.

5.2.4.3 Design Guide

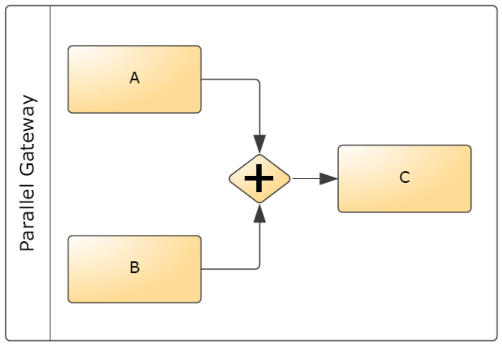
- The Event-based Gateway reflects a wait state in the process – no processing will occur until the occurrence of one out of multiple events.
- If a subsequent activity can be triggered by different events, merge the multiple alternative triggers with an XOR Join Gateway.
- If the event trigger is based on contextual information that is not easily captured by timer, message, or signal event types, a conditional intermediate event symbols should be used. The underlying contextual conditions can then be represented by a rule written outside of the BPMN diagram.

5.3 Join Patterns

5.3.1 Synchronized AND Join (AND-Join) [WCP-3]

Synchronized AND joins are used to merge multiple concurrent process execution paths into a single path. The AND gateway will wait until all upstream sequence flow have arrived before initiating the downstream sequence flow.

Figure 5-6: Synchronized AND Join Pattern



5.3.1.1 Semantics

Example in Figure 5-6: Upon encountering the AND gateway the process will halt until both Activity A and Activity B are completed. Once activities A and B are completed activity C can start.

All incoming sequence flows to an AND gateway have to signal the completion of their preceding activities or events before the process can continue.

5.3.1.2 Rationale

When an activity in a process requires data elements that are produced by multiple concurrent activities the AND-Gateway is necessary to ensure all required data elements are present before the process continues. It serves as a synchronization point in the process that signals that a subsequent activity is dependent on all activities that enter the merging AND Gateway.

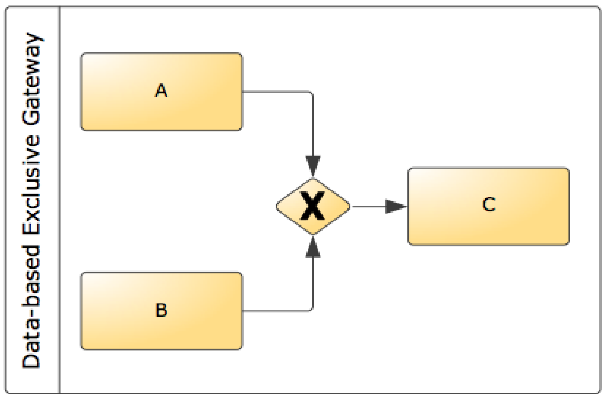
5.3.1.3 Design Guide

- Only use AND Joins to merge parallel paths that have been split with AND Splits.
- The imprudent use of AND Gateways is a common cause of deadlock situations where a process instance will not progress because the merging threads are alternative rather than concurrent.

5.3.2 Unsynchronized Join (XOR-Join) [WCP-5, WCP-8]

Unsynchronized XOR-Joins are used to merge alternative process execution paths into a single common path. The XOR gateway will react to each incoming sequence flow and initiate the downstream sequence flow

Figure 5-7: Unsynchronized XOR-Join Pattern



5.3.2.1 Semantics

At the Gateway the process will halt until either Activity A or Activity B is completed. Once either of these activities is completed Activity C can start.

If both activity A and B are completed the process may create multiple instances of downstream activities (i.e. Activity C in the figure above). This may create unwanted side-effects, e.g., the results of the first instance of Activity C might be lost or duplicate records may be created.

5.3.2.2 Rationale

The XOR Join is used to merge alternative control flows into a common thread. It is commonly used to end a number of alternative execution paths that have been selected through an XOR Split or an Event-based Gateway.

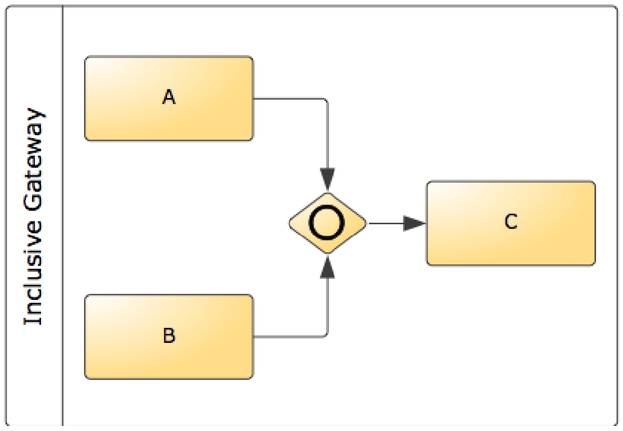
5.3.2.3 Design Guide

- Only use XOR Joins to merge alternative paths that have been split with XOR Splits.
- Never use an XOR Join to merge parallel process paths – this might lead to an unwanted duplication of downstream activities.
- The number of threads split and merged should match, if at all possible.

5.3.3 Synchronized OR Join (OR-Join) [WCP-7, WCP-9]

The synchronized OR-Join pattern is used to merge multiple process threads into a single common thread. The incoming process threads may contain parallel and/or alternative threads, and the OR gateway will evaluate merge conditions to determine whether the process can proceed. Generally, the OR join is a synchronizing merge operation in that it requires all incoming process threads to have completed before the downstream sequence flow can be initiated. The precise semantics of the OR join are complex. For instance, an executable implementation of the OR-join requires visibility into the upstream processing threads to determine whether or not a particular thread is still "alive".

Figure 5-8: Synchronized OR-Join Pattern



5.3.3.1 Semantics

Upon reaching the Gateway the process will halt until at least one incoming sequence flow (from Activity A or Activity B)is completed and it is apparent that no additional sequence flows will complete. At this point the underlying conditions of the OR Join are evaluated and the gateway determines if the downstream Activity C can be initiated.

At least one incoming sequence flow has to complete before the process continues.

The OR Join typically mirrors the condition of upstream OR Splits.

5.3.3.2 Rationale

OR Joins are used to synchronize *n* out of *m* parallel process threads. While useful at the analysis level, OR Joins are difficult to implement in process execution platforms because the number of arriving tokens is only known at runtime (i.e., when a process instance is being executed). An OR Join should always have a corresponding OR Split. Not all BPMS thus support the OR Gateway.

5.3.3.3 Design Guide

- Only use OR Joins to merge parallel paths that have been split with OR Splits

6 High-Level BPMN Design Patterns

The elementary design patterns discussed in section 5 can be used in a variety of situations. However, in certain situations it is desirable to have a solution pattern for a common semantic context. The Joint Capability Areas (JCAs) provide a framework for such patterns. In the following section we focus on patterns that provide solutions to common modeling issues surrounding JCAs. These patterns cover aspects such as monitoring, mediation, and collaboration.

While the low-level patterns are a set of *normative* modeling elements that restrict the use of BPMN for architects, the high-level patterns are *recommended* solutions for common modeling problems. By nature, the high level patterns are not exhaustive, thus we encourage the submission of additional patterns by architects that have developed them.

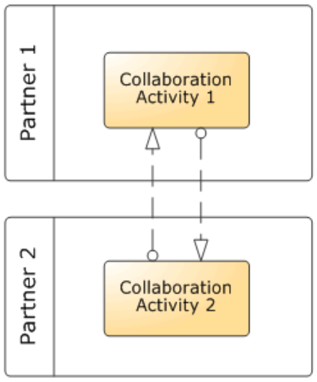
Each high-level pattern can be customized to fit the purpose of the modeler, within the constraints of the primitives and low-level patterns described in the previous section. For example, the collaboration patterns can be extended to include more than two collaborating parties. The monitoring pattern can be altered to accommodate continuous monitoring that is bounded by the receipt of a particular message rather than the expiration of a timer, etc.

6.1 Collaboration Patterns

6.1.1 Abstract Collaboration

The Abstract Collaboration pattern is used when two (or more) parties jointly work toward a specific outcome. While the outcome is defined, the process to arrive at this outcome is typically unstructured or unregulated. Abstract Collaboration is indicated by two activities in separate pools that are linked through message flows. If the collaborators are part of the same sphere of control, collaboration can be modeled by placing the collaborative activity in a separate lane that represents a group of all involved parties. For example, in the example shown in Figure 6-1, if Partner 1 and Partner 2 were part of the same organization they would be represented by two swimlanes within the same pool. To represent collaboration a third swimlane titled "Partners 1 & 2" could be added to the pool and "Collaboration Activity" would be placed within this swimlane.

Figure 6-1: Abstract Collaboration Pattern

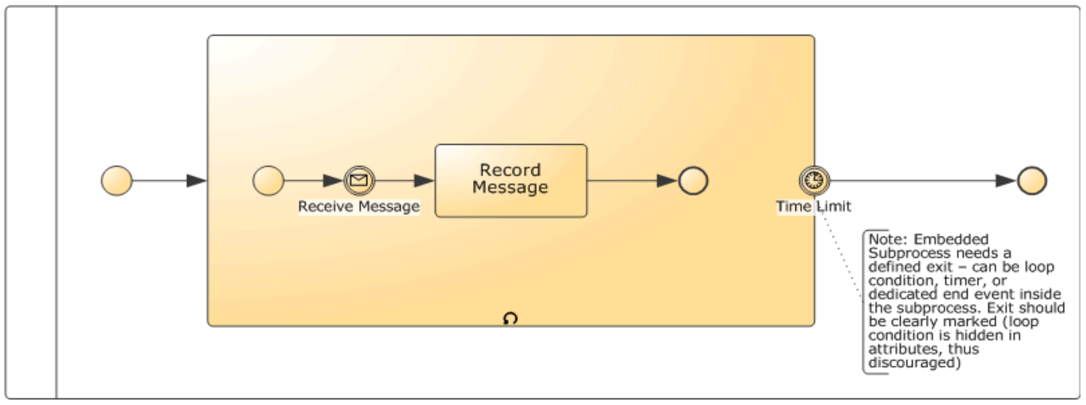


If two separate swimlanes are involved in the collaboration pattern, both activities should be labeled with the same name prefix ("Collaboration Activity" in the example above)to indicate the collaborative overlap of the activities.

6.1.2 Monitoring

The Monitoring pattern is used when content that arrives on a messaging channel (Wiki/email/blog/radio etc.) is recorded (and possibly summarized). Monitoring technically is a state rather than an activity in that it is a period of time during which reactive a system reacts to external events (i.e. the incoming messages) rather than actively affecting the external system. In BPMN this can be resolved by using an attached timer event as the exit from an activity. This pattern is based on the state-like activity pattern described in section 6.4.1.

Figure 6-2: Monitoring Pattern

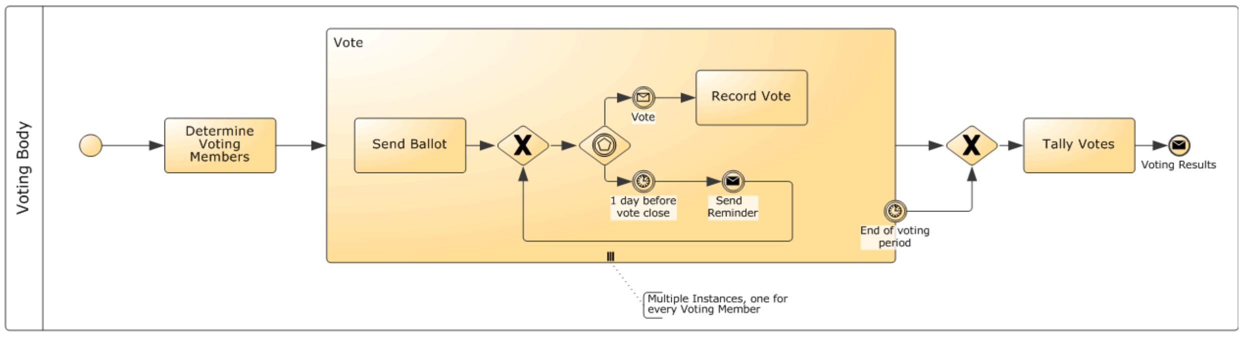

Receive Message
Record Message
Time Limit
Note: Embedded Subprocess needs a defined exit – can be loop condition, timer, or dedicated end event inside the subprocess. Exit should be clearly marked (loop condition is hidden in attributes, thus discouraged)

In the above pattern an expanded subprocess is repeated until a time limit is reached. With each iteration the embedded subprocess will start and immediately wait for the receipt of a message. Once a message is received it will be recorded and the iteration of the subprocess completes, to be succeeded by a new iteration. Once the time limit is reached the subprocess will terminate and the embedding process will continue.

6.1.3 Voting

A frequent collaboration pattern is a voting situation where multiple members of a governing body cast votes that determine the approval or rejection of an issue/document/solution. A vote will be open for a voting period, but can be closed when all eligible voters have cast their ballot. The following BPMN diagram illustrates the typical voting process.

Figure 6-3: Voting Pattern

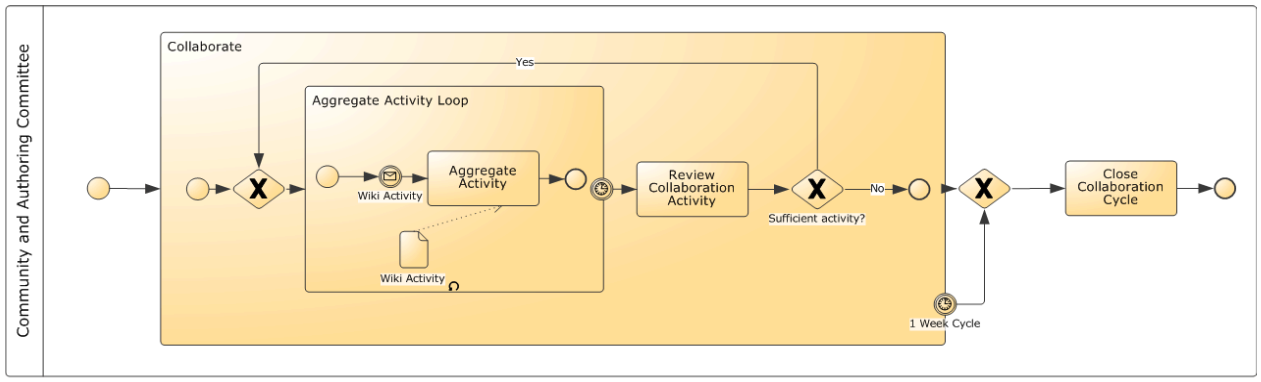


In an initial activity the list of eligible voting members is determined. The following activity is performed multiple times in parallel, once for every voting member that has been determined in the previous step. It follows the state-like activity pattern described in section 6.4.1.A ballot is sent to each member. Subsequently, one of two events can occur: A predefined point in time before the end of the voting period is reached (a timer expires one day before vote close in the example above), or the voter casts his/her ballot. If the vote is cast it is recorded. If the timer expires before the vote is cast a reminder is sent to the voting member. The process continues when all votes have been recorded or at the end of the voting period, regardless of the total number of votes cast. In the next step the votes are tallied.

6.1.4 Collaborative Editing

The Collaborative Editing pattern is used in situations where multiple members of an editing/authoring committee contribute content and comments on a document/model/architecture during an editing/collaboration phase. It is an example of a refined monitoring pattern and thus can provide an illustration how several generic patterns can be refined to form a more specific pattern.

Figure 6-4: Collaborative Editing



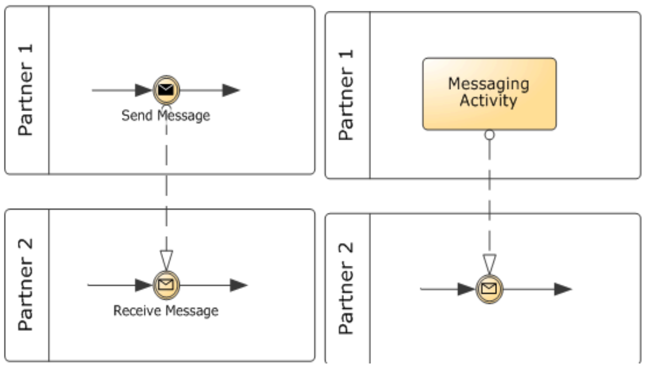
At the heart of this pattern is the monitoring loop described in section 6.1.2, which in turn is is based on the state-like activity pattern (described in section 6.4.1). This loop is triggered by user activity on a collaborative medium (a Wiki in the example, but this could also be email, a blog, a whiteboard or another medium). New activity is summarized in the “Aggregate Activity” step until a timer expires (e.g. a weekly timer). Once the timer expires the summarized collaboration activity is reviewed. If there is a sufficient level of activity (e.g. if there is a difference of opinion, or a certain amount of new content that has been contributed) the collaboration cycle begins again. If there is insufficient activity, or if a predefined timeframe expires (1 week in the example) the collaboration activity ends and the collaboration cycle is closed.

6.2 Messaging Patterns

6.2.1 Unidirectional Messaging

In a unidirectional messaging pattern the sending partner will send a message to a receiving partner without preparing for, expecting, or processing a response. A message in BPMN can have only one sender and recipient, thus it is a point-to-point pattern. If a message is to be sent to multiple recipients a signal event should be used.

Figure 6-5: Unidirectional Messaging

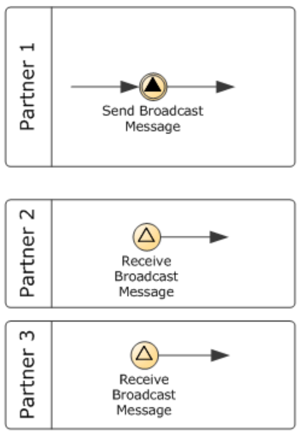


Note: BPMN provides a distinct symbol for sending messages (message throwing event). It is left to the discretion of the modeler to decide whether the sending activity is complex enough to warrant the use of the activity symbol, or whether the use of a throwing message event will suffice.

6.2.2 Broadcast Messaging

In some processes multiple messages may result from the occurrence of an event. In BPMN a message throwing event can have only one recipient (see One-Directional Messaging above). If the same message is to be sent to multiple recipients, a signal event should be used. This indicates a broadcast of the message and an unlimited number of subscribers can react to the receipt of the message.

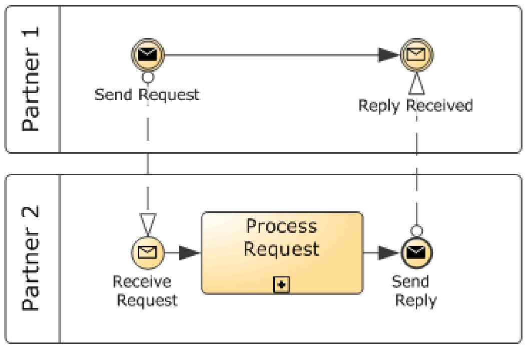
Figure 6-6: Broadcast Messaging



6.2.3 Synchronous Request/Response

The request/response pattern is a common messaging pattern where a requester sends a message to a recipient and waits for a response from the same recipient. Request/Response patterns are often blocking, in that the requester will wait for the reply (synchronous messaging).

Figure 6-7: Synchronous Request Response



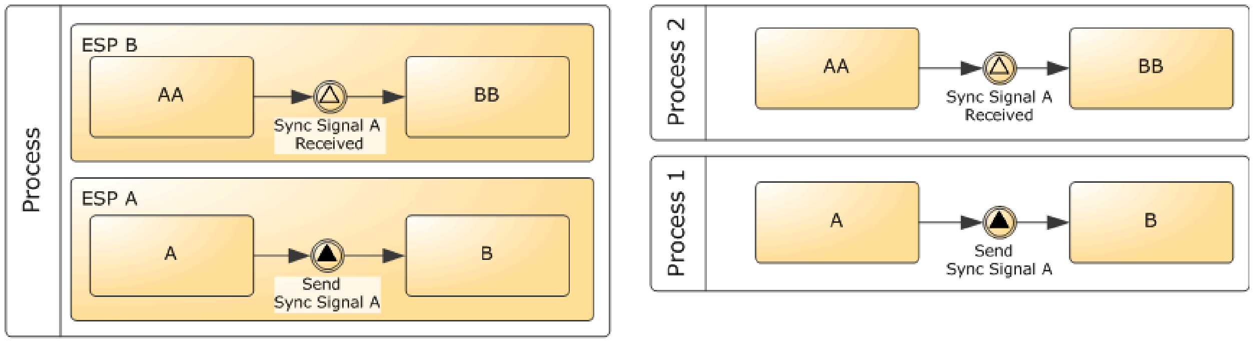
BPMN 1.2 provides an event type "Message" to reflect point-to-point messages. Filled envelopes indicate throwing events (i.e. those sending a message); outlined envelopes indicate catching events (i.e. those reacting to the arrival of a message).

Note that the request/response pattern does not prescribe the medium of communication (radio, email, fax, etc.) and the security of the communication channel (authentication, encryption etc.). These facets would be described at a lower level of granularity.

6.2.4 Milestone Synchronization [WCP-18]

In certain situations the continuation of a process will depend on the progress of a different process. In BPMN this can be resolved through a message or a signal event. A signal event is appropriate if the milestone in the signaling process may serve as a "go-ahead signal" for multiple other processes. This inter-process synchronization can be accomplished at multiple levels. If two distinct processes need to be synchronized the signal event will broadcast across pools. If two activities within the same process need to be synchronized a signal event can be used within the same process level or embedded subprocesses to send a synchronization signal.

Figure 6-8: Milestone Synchronization

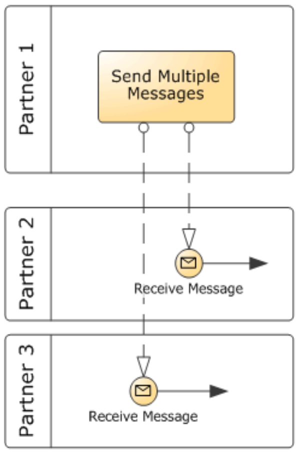


In the example above, activity BB may be expensive or risky to execute. It requires as input data from activity AA, but it should only be performed after activity A has been successfully completed. The example on the right shows the solution pattern if multiple distinct processes in different pools are involved, while the example on the left shows the same semantics using embedded subprocesses within one pool. Process 1 (Embedded Sub Process A) will broadcast a sync signal once activity A has been completed. Process 2 (Embedded Sub Process B) will halt after activity AA has been completed until the signal from process 1 (Embedded Sub Process A) has been received.

6.2.5 Multiple Messages from Event

If different messages are sent to different recipients an activity with multiple outgoing message flows can be used to represent the concurrent sending of multiple messages. Note that we interpret multiple outgoing message flows from an activity to be *concurrent*, i.e. all messages are sent, independent of the processing context.

Figure 6-9: Multiple Messages

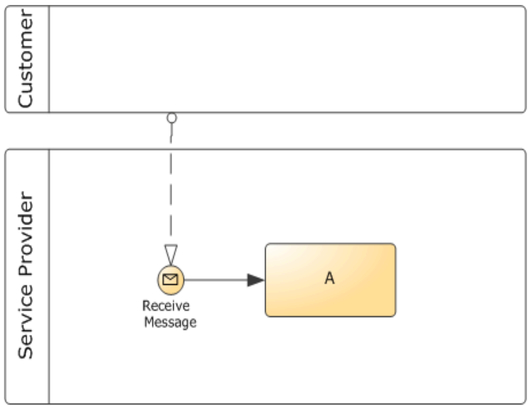


If different sets of messages are sent under different conditions these conditions should be made explicit and differentiated using *Exclusive* or *Inclusive Choice* patterns.

6.2.6 External Process Trigger

Only scheduled processes can be represented in a self-contained fashion, they are triggered by the expiration of a timer. In many situations, however, processing is the result of some external event or trigger. In a BPMN diagram this can be achieved by using a catching message event as the process trigger. In some situations it may be desirable to include the external party that triggers the process in the overall diagram.

Figure 6-10: External Process Trigger

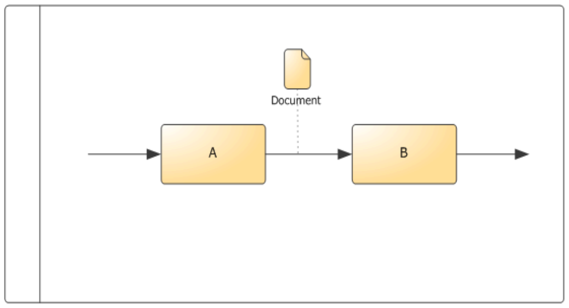


In cases where an external process trigger should be included in the diagram the triggering party should be represented as a separate pool above the core pool (the triggered process).

6.2.7 Explicit Document/Data Flow

A sequence flow between two activities indicates a precedence/successor relationship. This relationship can be caused by temporal, performer, or resource (i.e. data) dependencies. In the case of data dependencies it may be desirable to visualize the document or data element that is passed between two activities. This can be accomplished by linking a BPMN data object via an undirected Association with the sequence flow between two activities.

Figure 6-11: Explicit Document/Data Flow

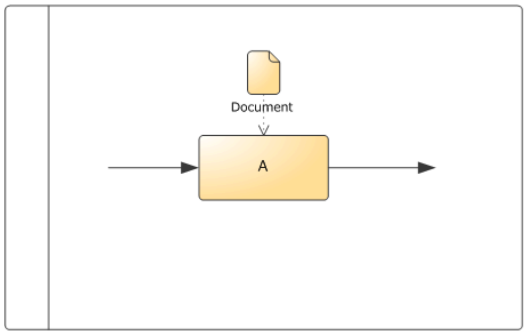


Note that data objects are embellishments of the process model but do not contain execution semantics, i.e. this use of the data object is for illustration purposes only. In BPMN data input/output relationships are handled via input sets and output sets that are associated with each activity object. These input and output sets are multi-valued attributes that are not graphically represented in the diagram. In addition, the OV-3 Operational Resource Flow Exchange Matrix is designed to provide detailed information about the input and output data elements of each activity.

6.2.8 Supplemental Document/Data Flow

In some process activities there may be documents that can be used to complete the activity, but that are not an essential precondition for the start (or the completion) of the activity. Common examples are manuals, guidelines, or instructions. The existence of these documents is not the result of the process itself, thus they are not a normal part of the sequence flow between activities that moves the results of one activity to the next. Document elements can be used to indicate supplemental documents that are input to an activity.

Figure 6-12: Supplemental Document Flow

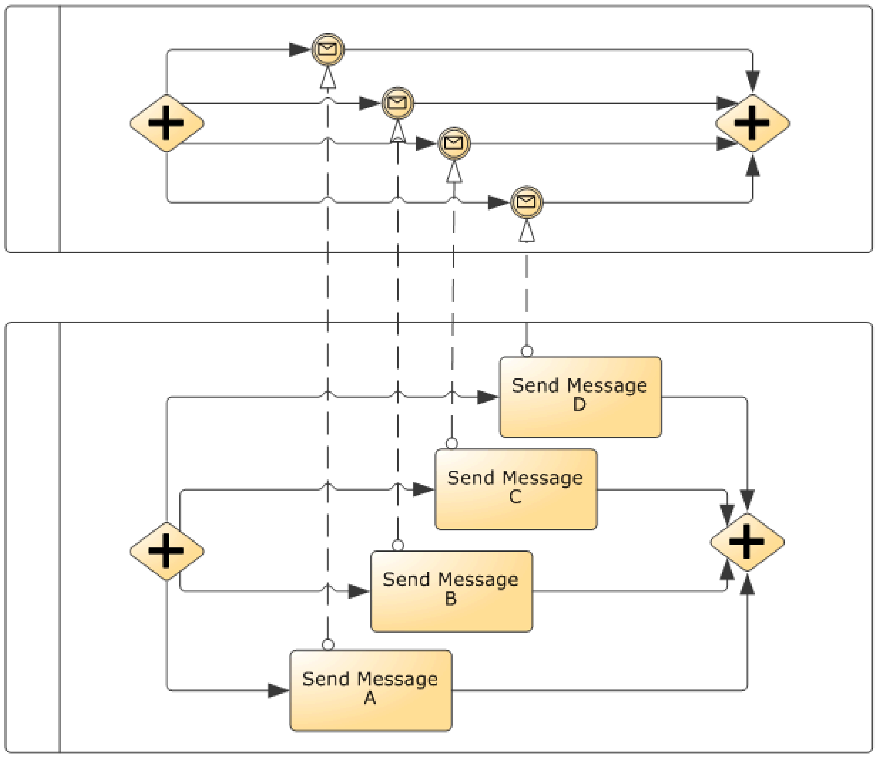


Supplemental documents can be the output of an activity if they are used for governance and compliance purposes (such as a processing log), but are not consumed by any of the subsequent activities in the process. In these cases a document element can be connected to an activity with an arrow pointing from the activity to the document.

6.3 Mediation Patterns

Mediation is the aggregation/disaggregation of information between a sender and a recipient. In the example below four pieces of data are sent from the bottom pool to the top pool. Depending on the configuration of the communication medium the number of sending and receiving events may differ between the two pools. A consistent representation of mediation thus requires the modeler to explicitly represent the communication channel in the diagram as a separate pool.

Figure 6-13: Mediation Example



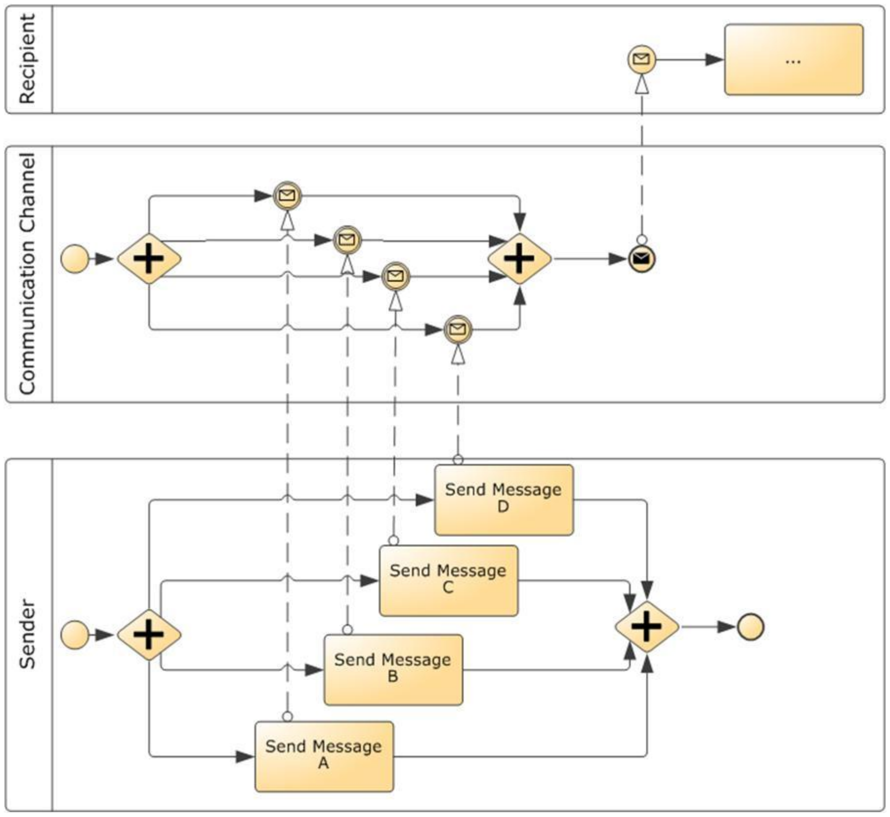
The following aspects of both the multiplexing and de-multiplexing patterns discussed below should be stressed:

- The communication channel can be a technical device, a software, or a human. The use of a separate pool is necessary to retain the messaging character of these patterns.
- Multiplexing and de-multiplexing can be extended to include an arbitrary number of senders and/or recipients. For example, it is possible in the above scenario to package the messages A through D in two separate messages that are sent to two different recipients.
- The communication channel pool can be used to model message transformation and conditional routing logic (e.g., forwarding of partial messages in the absence of other information).

6.3.1 Multiplexing

The multiplexing pattern takes an arbitrary number of incoming messages and converts them into a single outgoing message for the recipient. This pattern is useful to represent a bandwidth limited situation where the management overhead associated with each message may contribute to a bottleneck situation.

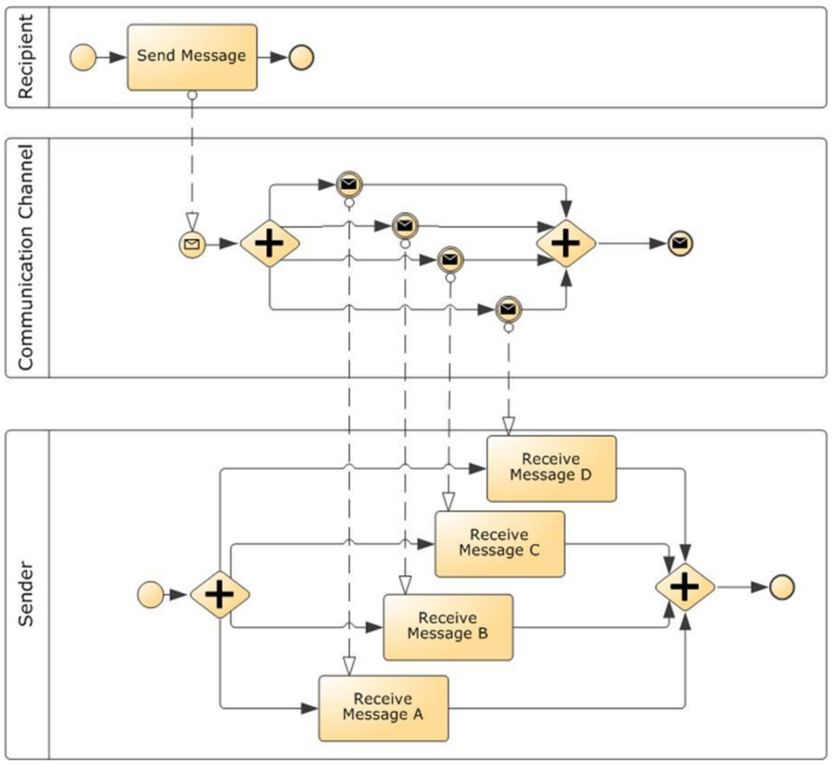
Figure 6-14: Multiplexing Pattern



6.3.2 De-Multiplexing

The de-multiplexing pattern represents the inverse of the multiplexing pattern. In this situation a single message is broken into separate individual messages that are sent to one (or multiple) recipients.

Figure 6-15: De-Multiplexing Pattern

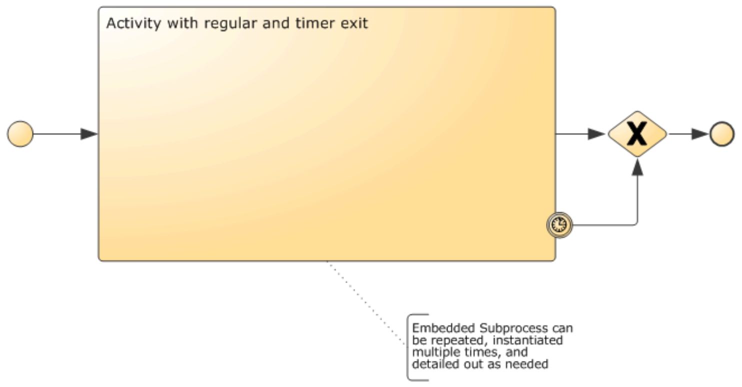


6.4 Miscellaneous Patterns

6.4.1 State-like Activity

Sometimes a modeler may need to represent an activity that will complete upon the expiration of a timer or the manual completion of the activity. To represent this situation the modeler can attach a timer event to the boundary of the activity – once the timer expires the execution of the activity will cease and the sequence flow originating from the timer event will be followed. If the activity completes prior to the expiration of the timer a regular sequence flow from the activity will be followed. The sequence flows from the timer event and the regular sequence flow may be merged using an unsynchronized join (XOR Join), but can be kept separately if different downstream activities result from the different modes of activity completion.

Figure 6-16: State-like Activity Pattern


Activity with regular and timer exit
X
Embedded Subprocess can be repeated, instantiated multiple times, and detailed out as needed

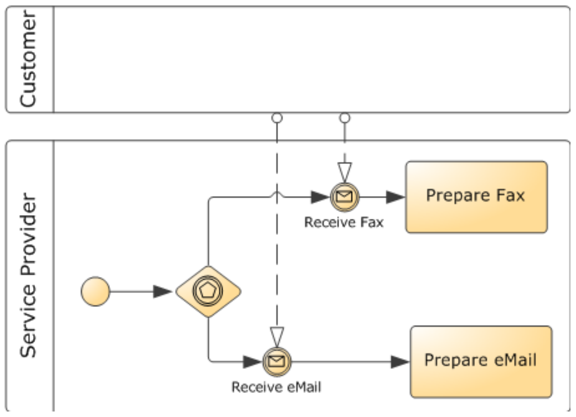
6.4.2 Multiple Start Events

Some processes may be triggered through multiple channels – for example, a process may be started by fax or by email. Each trigger may require specific activities before the (common) main process can commence. There are several solutions to this scenario.

The simplest solution is to split the process in question into two parts: Multiple *feeder processes* that react to the various event combinations, and one common main process that is triggered by the completion of any of the feeder processes.

BPMN provides a gateway type to handle mutually exclusive events, the event-driven gateway.

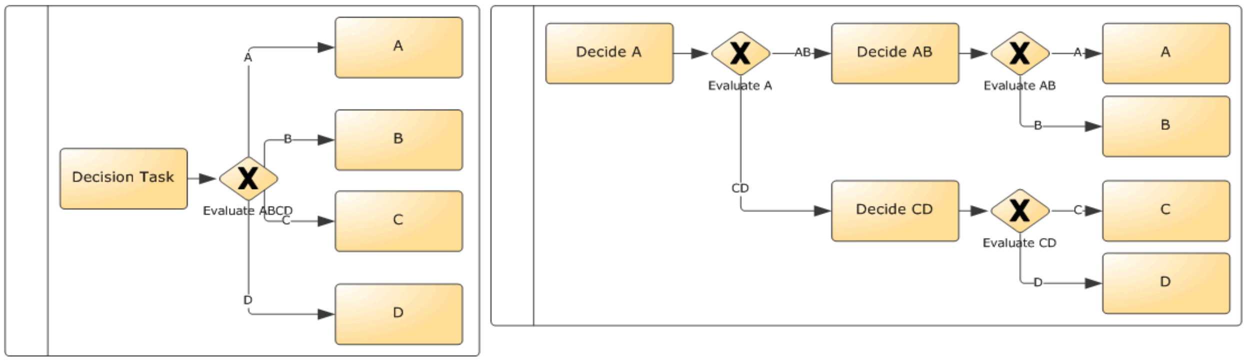
Figure 6-17: Multiple Start Events



6.4.3 Multi-Step Decisions

Multi-Step Decisions are common in processes that exhibit a decision-tree-like logic. In these cases multiple criteria of a decision table are evaluated one at a time either by the same or by different roles. The low-level patterns *Exclusive Choice* and *Inclusive Choice* can be used to reflect the decision logic. However, the gateways themselves do not make decisions; they react to the outcome of a decision by testing the value of a result variable. A decision-making activity is necessary before an Exclusive or Inclusive Choice pattern.

Figure 6-18: Multi-Step Decisions



In the example on the left, the *Decision Task* activity is sufficient to decide which of the subsequent four activities should be performed. In the example on the right the decision logic is broken down into individual criteria (A, AB, and CD) that are evaluated by three distinct decision-making activities.

Multiple decision-making activities are required if:

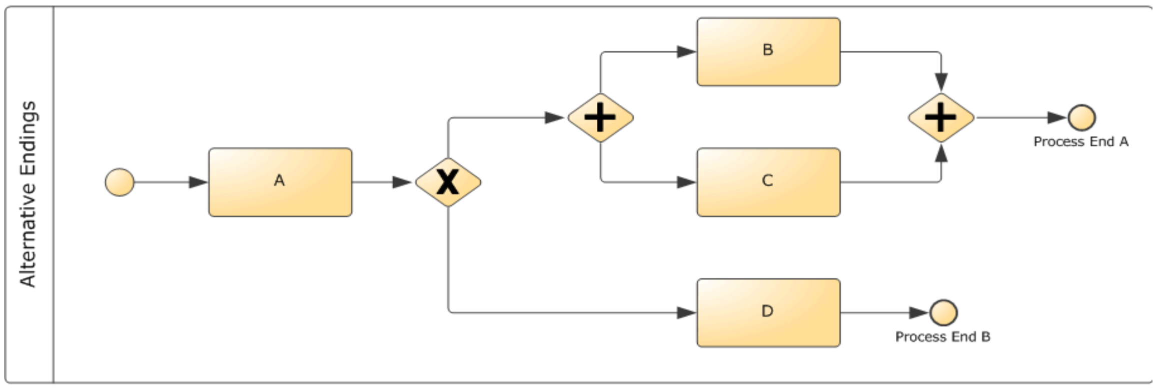
- Different decision-makers evaluate the different criteria, and/or
- The results of one decision serve as the input for the next decision, and/or
- The decision logic should be visualized in the process

The modeler has to trade off the explicit rendering of the decision logic with the overall complexity of the process model. If the decision logic will be externalized, e.g. by using a rules engine it may be sufficient to model just one complex decision-making activity. If multiple human participants are involved in the decision-making process it is advisable to explicitly model the decision logic.

6.4.4 Multiple End Events

In some cases a process may have multiple possible outcomes. Multiple BPMN end events can be used to depict different exit points from a process. Note that once such an exit point has been reached no other processing in the same pool is permitted, i.e. end events cannot be used to terminate one of multiple parallel process threads. Instead these threads should be merged to one common exit point.

Figure 6-19: Multiple End Events

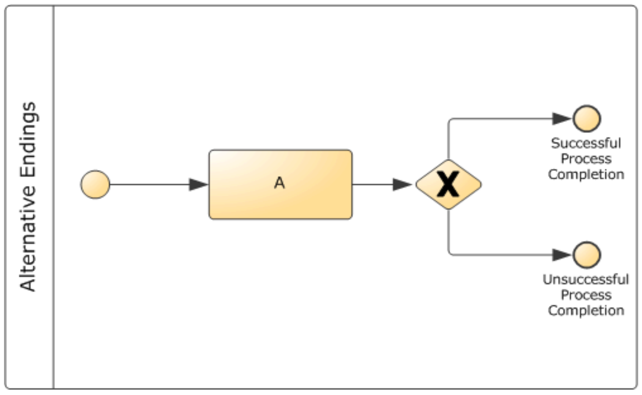


The figure above shows a process with two alternative endings (Process End A and Process End B). This is a legal pattern as long as no other thread of the process is active when either End A or End B is reached.

6.4.5 Negative Process Outcome

In some instances it may be desirable to visualize a negative process outcome. A negative process outcome does not indicate that the process has failed (i.e. the processing itself did not conclude properly), but that the result of the process does not meet the expectations of a "best-case scenario", for example the disapproval of a mission.

Figure 6-20: Negative Process Outcome


Alternative Endings
A
X
Successful Process Completion
Unsuccessful Process Completion

The figure above shows an alternative ending pattern where activity A has two possible outcomes: A successful and an unsuccessful outcome. In both cases activity A is successfully completed, but the following Exclusive Choice Gateway will evaluate the outcome and choose the required exit point.

Note: Multiple end events *must not* directly originate from a single activity, as this would indicate that all end points are simultaneously reached after the completion of the activity. In cases where multiple end events are required these must always be preceded by the appropriate gateway to indicate their logical relationship (mutually exclusive vs. concurrent).

7 References

BEA Development Methodology Version 3.0. BEA Development Team. US DoD Business Transformation Agency, March 14, 2008

Business Process Modeling Notation. V1.2 OMG Available Specification, OMG formal/2009-01-03, 316 pages. Available at http://www.omg.org/spec/BPMN/1.2

DoD Architecture Framework Version 2.0 Volume I: Introduction, Overview, and Concepts. Manager's Guide. DoDAF V2.0 Development Team. US OSD-NII, May 28, 2009

DoD Architecture Framework Version 2.0 Volume II: Architecture Data and Models. Architect's Guide. DoDAF V2.0 Development Team. US OSD-NII, May 28, 2009

Workflow Control-Flow Patterns: A Revised View. N. Russell, A.H.M. ter Hofstede, W.M.P. van der Aalst, and N. Mulyar. BPM Center Report BPM-06-22, BPMcenter.org, Eindhoven, NL and Brisbane, Australia 2006.

Workflow Patterns. W.M.P van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Distributed and Parallel Databases, 14(3), pages 5-51, July 2003.

Guidelines of Business Process Modeling. Becker, J., Rosemann, M. & von Uthmann, C.; In: Business Process Management. Models, Techniques, and Empirical Studies, (Eds, van der Aalst, W.M.P., Desel, J. & Oberweis, A.) Springer, Berlin, Germany, 2000, pp. 30-49.

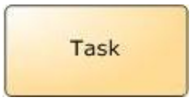
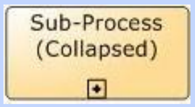
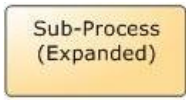
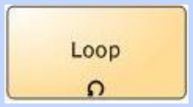
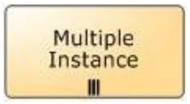
What is an Ontology? Guarino, N.; Oberle, D.; Staab, S., In: S. Staab & R. Studer. Handbook on Ontologies. 2nd revised edition. Springer, 2009.

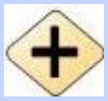
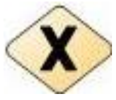
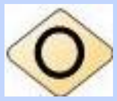
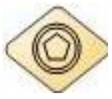
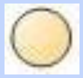
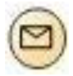
What Makes a Good Data Model? Evaluating the Quality of Entity Relationship Models. Moody, D.L.; Shanks, S.; In: Loucopoulos, P. (Eds.): Entity-Relationship Approach - ER'94. Business Modelling and Re-Engineering. 13th International Conference on the Entity-Relationship Approach. Berlin, Heidelberg etc.: Springer 1994, pp. 94-111.

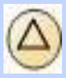
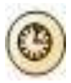
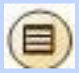
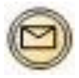
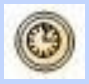
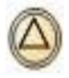
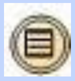
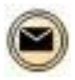
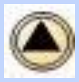
8 Appendix A: BPMN Primitives

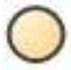
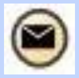
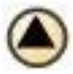
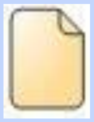
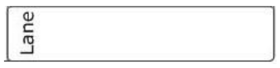
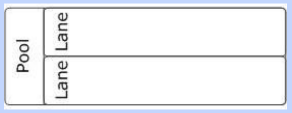
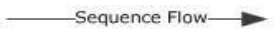
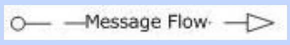
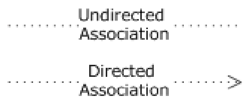
The table below identifies the initial set of BPMN primitive modeling elements relevant in the architectural view OV-6C, and describes their relationship to core elements of the DoDAF V2.0 Meta Model (DM2).

Table 8-1 Primitive BPMN Modeling Elements for OV-6C

Primitive	OV-6C Semantics	BPMN Symbol	DM2 Element
Task	A Task is an atomic activity that is included within a Process. A Task is used when the work in the Process is not broken down to a finer level of Process Model detail. Generally, an end-user and/or an application are used to perform the Task when it is executed.		Activity
Sub-Process	A Sub-Process is a Process that is included within another Process. A Sub-Process shares the same shape as the Task, which is a rectangle that has rounded corners. A Sub-Process is marked by a [+] marker at the bottom of the symbol.		Activity
Expanded Sub-Processes	An expanded Sub-Process is a Sub-Process that shows its details within the view of the Process in which it is contained. An expanded Sub-Process is not marked separately, as it can be identified by the BPMN symbols contained therein.		Activity
Loop Marker	A loop marker indicates that a task will repeat depending on some condition set at the attribute level of the task. A loop marker can be used with a Sub-Process as well.		Activity
Multiple Instance Marker	A multiple instance marker indicates that multiple concurrent instances of a task (or sub-process) will be created at run-time. How many of these		Activity

Primitive	OV-6C Semantics	BPMN Symbol	DM2 Element
	instances need to complete before the task completes is defined at the attribute level of the Task (or Sub-Process).		
Parallel Gateway	A parallel gateway splits one process thread into multiple concurrent threads or merges multiple concurrent threads into one thread via a synchronized join (i.e. the outgoing sequence flow will only be activated one all incoming sequence flows have been activated).		Rule; RuleConstrainsActivity (for activities that flow out of the gateway);
Data-based Exclusive Gateway	A data-based exclusive gateway when used as a split routes the sequence flow from one incoming flow to exactly one of multiple outgoing flows. When used as a merge it will wait until one of the incoming sequence flows is traversed.		Rule; RuleConstrainsActivity (for activities that flow out of the gateway);
Inclusive Gateway	An inclusive gateway when used as a split activates one or more branches based on branching conditions. When used as a merge, it awaits all active incoming branches to complete.		Rule; RuleConstrainsActivity (for activities that flow out of the gateway);
Event-based Exclusive Gateway	An event-based exclusive gateway is always followed by catching events or by receive tasks. Sequence flow is routed to the subsequent event/task that is activated by the first event occurrence.		Rule; RuleConstrainsActivity (for activities that flow out of the gateway);
Start Event	A start event indicates the first node of a process.		Event
Message Start Event	A message start event indicates that the process will start once a particular message has been received.		Event

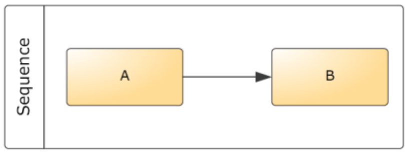
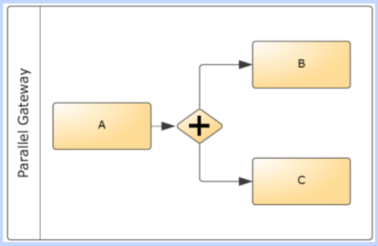
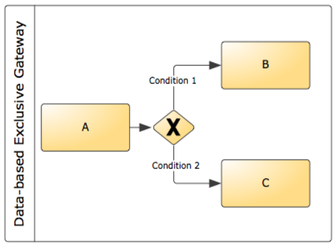
Primitive	OV-6C Semantics	BPMN Symbol	DM2 Element
Signal Start Event	A signal start event indicates that the process will start once a broadcast message has been observed.		Event
Timer Start Event	A timer start event indicates that the process will start at a specific time (or after a specific delay).		Event
Conditional Start Event	A conditional start event indicates that the process will start when a set of rules (conditions) evaluates to true.		Event
Intermediate Message Catching Event	An intermediate message catching event indicates that the execution of the process will halt until a specific message has been received.		Event
Intermediate Timer Catching Event	An intermediate timer catching event indicates that the execution of the process will halt until a specific message has been received.		Event
Intermediate Signal Catching Event	An intermediate signal catching event indicates that the execution of the process will halt until a broadcast signal has been observed.		Event
Intermediate Conditional Catching Event	An intermediate conditional catching event indicates that the execution of the process will halt until a specific set of rules evaluates to true.		Event
Intermediate Message Throwing Event	An intermediate message throwing event indicates that the process will send a message to a specific recipient at the point specified.		Event
Intermediate Signal Throwing	An intermediate signal throwing event indicates that the process will send a		Event

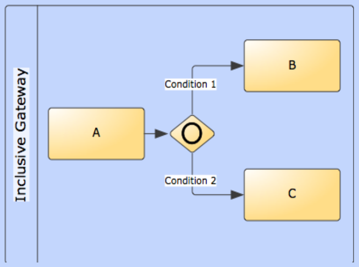
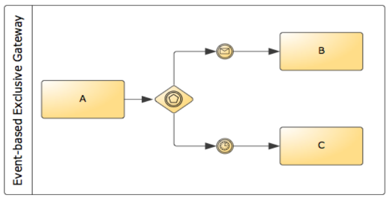
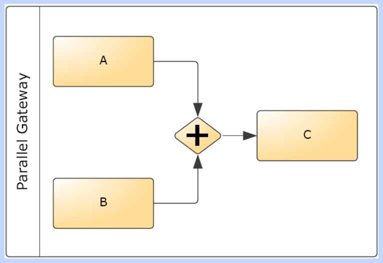
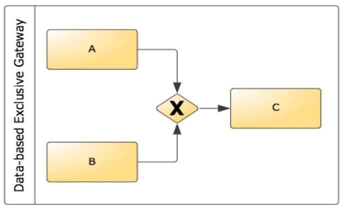
Primitive	OV-6C Semantics	BPMN Symbol	DM2 Element
Event	broadcast signal at the point specified.		
End Event	An end event indicates the last node of a process.		Event
End Message Event	An end message event sends (throws) a message at the end of the process.		Event
End Signal Event	An end signal event broadcasts (throws) a signal at the end of the process.		Event
Data Object	A data object represents data flowing through the process, indicating how documents, data, and other objects are used and updated during the process.		Data
Lane	Represents responsibilities for activities in a process.		Performer;ActivityPerformerOverlap (for activities in the lane)
Pool	Lanes subdivide pools (or other lanes) hierarchically.		Performer;ActivityPerformerOverlap (for activities in the lane)
Sequence Flow	Defines the execution order of activities. The Default Flow symbol indicates the standard execution path in the presence of OR or XOR splits.		ActivityBeforeAfter
Message Flow	Information flow across organizational boundaries.		ActivityResouceOverlap
Association	Attaching a data object with an undirected association to a sequence flow illustrates the hand-over of information between the activities involved. A directed association between a data object and an activity illustrates the availability of this data object for use in the execution of the activity.		ActivityResouceOverlap (DataObject isa Resource)

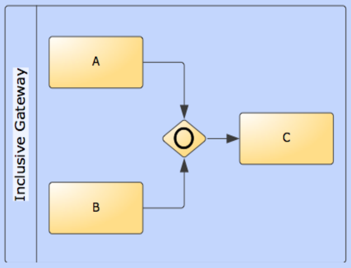
9 Appendix B: BPMN Low-Level Design Patterns

The BPMN Low-Level Design Patterns are derived from the underlying BPMN primitives as elementary design patterns. Additional guidance on rationale and modeling style using these 'low-level' patterns is provided in the Modeling Guide in section 5. Table 9-1 identifies the initial set of low-level BPMN design patterns and describes their relationship to core elements of the DoDAF V2.0 Meta Model (DM2).

Table 9-1 Derivative BPMN Modeling Patterns for OV-6C

Derivative	OV-6C Semantics	BPMN Pattern	DM2 Element
Sequence	Activity A must complete before Activity B can start. A sequence flow indicates a dependency between two activities, either through a shared performer or the exchange of data.		Activity; ActivityBeforeAfter
Parallel Split (AND)	After completion of Activity A both Activity B and Activity C can start independent of each other. Note the AND split does not imply that B and C have to occur at the same time.		Activity; ActivityBeforeAfter; Rule; RuleConstrainsActivity(downstream activities)
Exclusive Choice Split (XOR)	After completion of Activity A either Activity B or Activity C can start (but not both), depending on the truth values of Conditions 1 and 2. Only one outgoing sequence flow can evaluate to true.	 Data-based Exclusive Gateway A Condition 1 B Condition 2 C	Activity; ActivityBeforeAfter; Rule; RuleConstrainsActivity(downstream activities)

Derivative	OV-6C Semantics	BPMN Pattern	DM2 Element
Inclusive Choice Split (OR)	After completion of Activity A either Activity B or Activity C or both Activity B and C can start, depending on the truth values of Conditions 1 and 2. Multiple outgoing sequence flows can evaluate to true.	 Inclusive Gateway A B C Condition 1 Condition 2	Activity; ActivityBeforeAfter; Rule; RuleConstrainsActivity(downstream activities)
Event-Based Exclusive Choice Split (XOR)	After completion of Activity A the process will halt until one of the subsequent events occurs. Only one outgoing sequence flow from the Event-based Gateway can evaluate to true.		Activity; ActivityBeforeAfter; Event; Rule; RuleConstrainsActivity(downstream activities)
Synchronized Join (AND)	The process will halt at the AND join until both Activity A and Activity B are completed. All incoming sequence flows must send a token before the process continues.		Activity; ActivityBeforeAfter; Rule; RuleConstrainsActivity(downstream activities)
Unsynchronized Join (XOR)	The process will halt at the XORjoinuntil either Activity A or Activity B is completed. If multiple incoming sequence flows		Activity; ActivityBeforeAfter; Rule; RuleConstrainsActivity(downstream activities)

Derivative	OV-6C Semantics	BPMN Pattern	DM2 Element
	send tokens, the process *may* create multiple instances of downstream activities (dependent upon particular BPMN implementation) .		
Synchronized Join (OR)	The process will halt at the OR join until both Activity A and Activity B are completed. At least one incoming sequence flow has to send a token before the process continues.		Activity; ActivityBeforeAfter; Rule; RuleConstrainsActivity(downstream activities)