Date: June 2010

BPMN 2.0 by Example

Version 1.0 (non-normative)

OMG Document Number: dtc/2010-06-02

Standard document URL: http://www.omg.org/spec/BPMN/2.0/examples/PDF

Associated File: http://www.omg.org/spec/BPMN/2.0/examples/ZIP

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

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

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

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

GENERAL USE RESTRICTIONS

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

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

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

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c) (1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , MOF™ , OMG Interface Definition Language (IDL)™ , and OMG SysML™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

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

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

OMG's Issue Reporting Procedure

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

Table of Contents

Preface

OMG

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

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

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

OMG Specifications

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

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

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications

OMG Middleware Specifications

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

Platform Specific Model and Interface Specifications

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

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format,

may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

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

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

Typographical Conventions

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

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

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

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

Helvetica/Arial - 10 pt: Exceptions

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

1 Scope

This document provides a number of **BPMN 2.0** examples, which are non-executable **BPMN 2.0** models conforming to the Process Modeling Conformance class as defined in the the OMG specification **Business Process Model and Notation (BPMN) Version 2.0**. It is a non-normative document and its main goal is to assist in interpreting and implementing various aspects of the **BPMN 2.0** specification. The examples are provided in form of **Collaboration** diagrams, **Process** diagrams, and **Choreography** diagrams as well as machine-readable files using the **Extensible Markup Language (XML)**.

2 Conformance

As this is a non-normative document, an implementation, which claims conformance to any of the conformance classes defined in section 2 of the **BPMN 2.0** specification, is NOT REQUIRED to comply to statements made in this document. Furthermore, if there are any inconsistencies between the **BPMN 2.0** specification and this document, the statements of the **BPMN 2.0** specification always have precedence.

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

Business Process Model and Notation (BPMN) Version 2.0

- OMG, May 2010
  http://www.omg.org/spec/BPMN/2.0

RFC-2119

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

4 Additional Information

4.1 Changes to Adopted OMG Specifications

If there are any inconsistencies between the **BPMN 2.0** specification and this document, the statements of the **BPMN 2.0** specification are considered to be correct.

4.2 Acknowledgements

The following companies contributed to the content of this document:

- camunda services GmbH
- IBM Corp.
- PNA Group
- SAP AG
- Trisotech, Inc.

The following persons were members of the core teams that contributed to the content of this document:

- John Bulles (PNA Group)
- Jakob Freund (camunda services GmbH)
- Denis Gagné (Trisotech, Inc.)
- Falko Menge (camunda services GmbH)
- Matthias Kloppmann (IBM Corp.)
- Sjir Nijssen (PNA Group)
- Gerardo Navarro-Suarez (camunda services GmbH)
- Ivana Trickovic (SAP AG)
- Stephen A. White (IBM Corp.)

In addition, the following persons contributed valuable ideas and feedback that improved the content and the quality of this document:

- Joram Barrez (Alfresco)
- Mariano Benitez (Oracle)
- Conrad Bock (NIST)
- John Hall (Model Systems)
- Bernd Rücker (camunda services GmbH)

5 Small Examples introducing Core Concepts

This chapter introduces the core concepts of process modeling with BPMN. We will not explain every single symbol you can find in the diagrams, but show how process modeling in BPMN is basically done, how we can use pools and message flows for explicitly modeling collaborations between participants, and how we can (de-)compose process models with sub-processes and call activities. Those examples do not contain executable process models, but represent process models focusing on organizational aspects of business processes.

5.1 Shipment Process of a Hardware Retailer

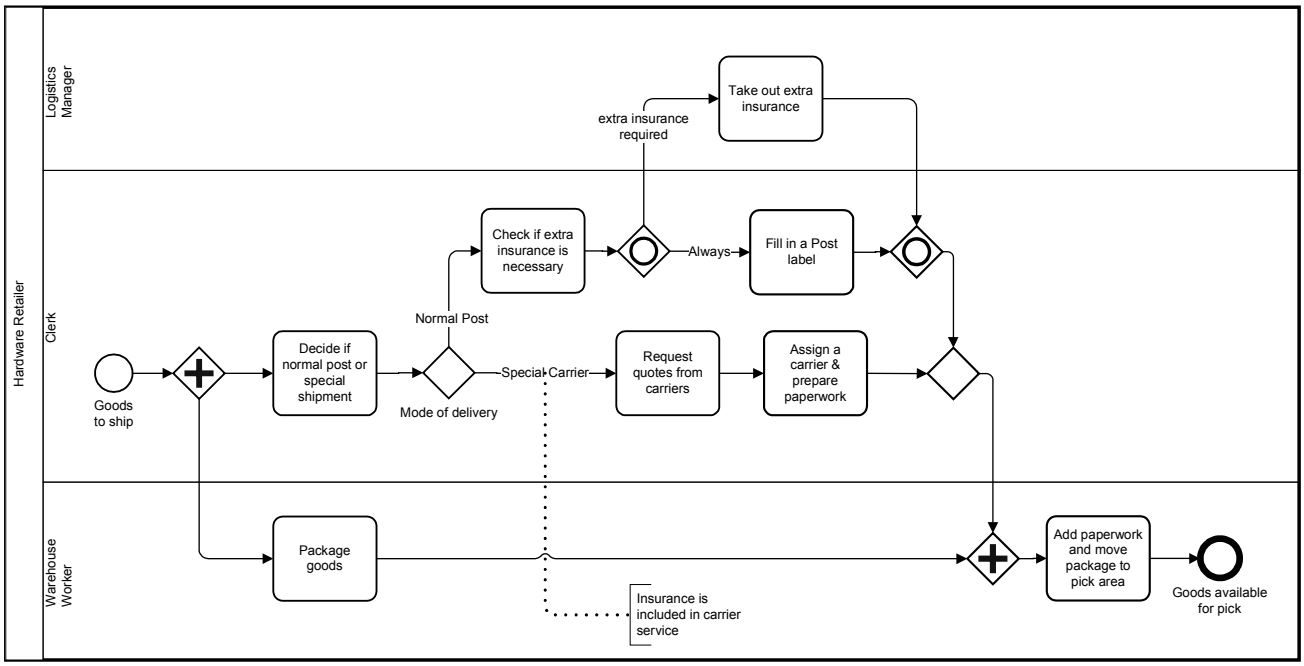


Figure 5.1: Shipment Process of a hardware retailer

In Figure 5.1 you can find the preparing steps a hardware retailer has to fulfill before the ordered goods can actually be shipped to the customer.

In this example, we used only one pool and different lanes for the people involved in this process, which automatically means that we blank out the communication between those people: We just assume that they are communicating with each other somehow. If we had a process engine driving this process, that engine would assign user tasks and therefore be responsible for the communication between those people. If we do not have such a process engine, but want to model the communication between the people involved explicitly, we would have to use a collaboration diagram as in the next chapter.

The plain start event “goods to ship” indicates that this preparation should be done now. Right after the instantiation of the process, there are two things done in parallel, as the parallel gateway indicates: While the clerk has to decide whether this is a normal postal or a special shipment (we do not define the criteria how to decide this inside the process model), the warehouse worker can already start packaging the goods. This clerk's task, which is followed by the exclusive gateway “mode of delivery”, is a good example for clarifying the recommended usage of a gateway: The gateway is not responsible for the decision whether this is a special or a postal shipment. Instead, this decision is undertaken in the activity before. The gateway only works as a router, which is based on the result of the previous task, and provides alternative paths. A task represents an actual unit of work, while a gateway is only routing the sequence flow.

This gateway is called “exclusive”, because only one of the following two branches can be traversed: If we need a special shipment, the clerk requests quotes from different carriers, then assigns a carrier and prepares the paperwork. But if a normal post shipment is fine, the clerk needs to check if an extra insurance is necessary. If that extra insurance is

required, the logistics manager has to take out that insurance. In any case, the clerk has to fill in a postal label for the shipment. For this scenario, the shown inclusive gateway is helpful, because we can show that one branch is always taken, while the other one only if the extra insurance is required, but IF it is taken, this can happen in parallel to the first branch. Because of this parallelism, we need the synchronizing inclusive gateway right behind “Fill in a Post label” and “Take out extra insurance”. In this scenario, the inclusive gateway will always wait for “Fill in a Post label” to be completed, because that is always started. If an extra insurance was required, the inclusive gateway will also wait for “Take out extra insurance” to be finished. Furthermore, we also need the synchronizing parallel gateway before the last task “add paperwork and move package to pick area”, because we want to make sure that everything has been fulfilled before the last task is executed.

5.2 The Pizza Collaboration

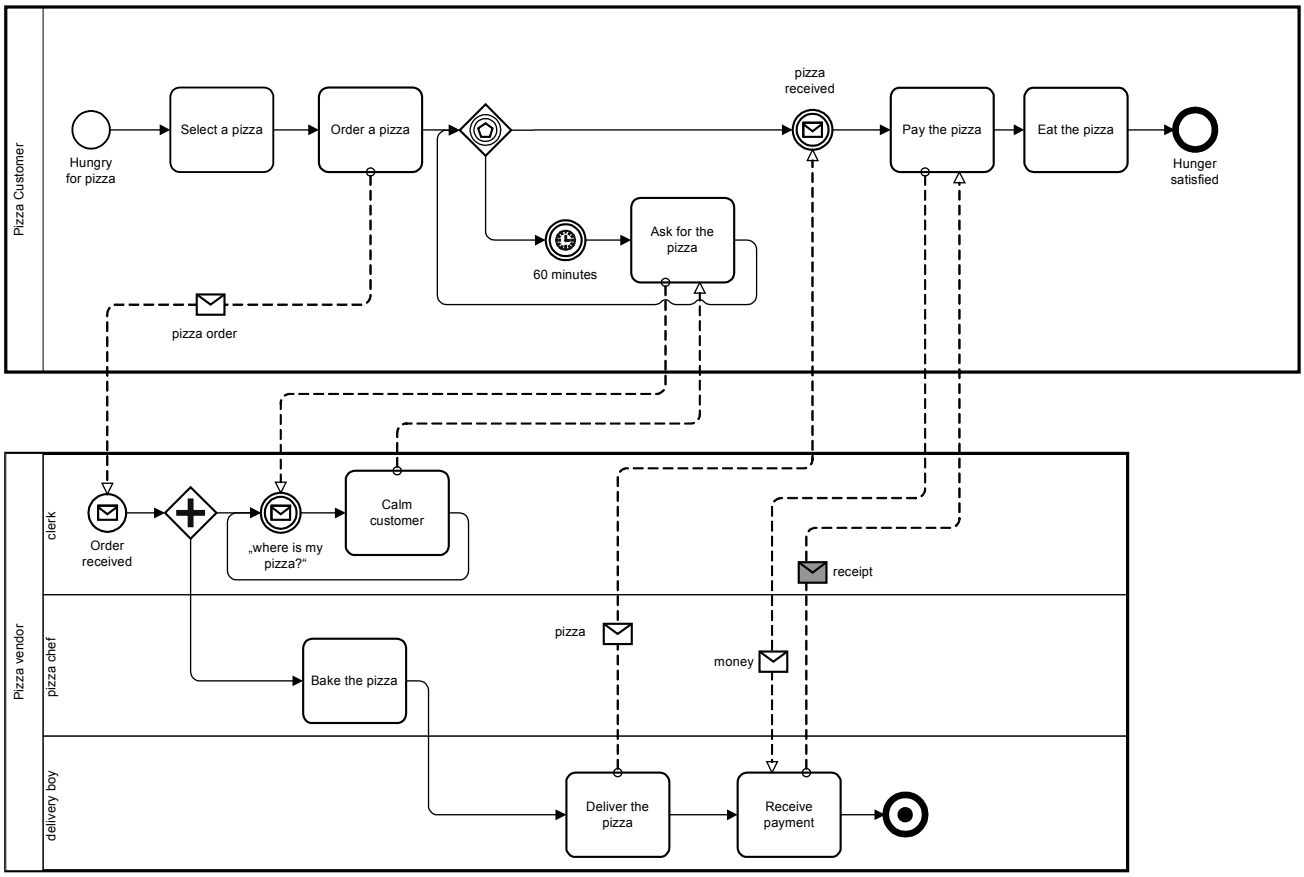


Figure 5.2: Ordering and delivering pizza

This example is about Business-To-Business-Collaboration. Because we want to model the interaction between a pizza customer and the vendor explicitly, we have classified them as “participants”, therefore providing them with dedicated pools. Please note that there is no default semantics in this type of modeling, which means you can model collaboration diagrams to show the interaction between business partners, but also zoom into one company, modeling the interaction between different departments, teams or even single workers and software systems in collaboration diagrams. It is totally up to the purpose of the model and therefore a decision the modeler has to make, whether a collaboration diagram with different pools is useful, or whether one should stick to one pool with different lanes, as shown in the previous chapter.

If we step through the diagram, we should start with the pizza customer, who has noticed her stomach growling. The customer therefore selects a pizza and orders it. After that, the customer waits for the pizza to be delivered. The event based gateway after the task “order a pizza” indicates that the customer actually waits for two different events that could happen next: Either the pizza is delivered, as indicated with the following message event, or there is no delivery for 60

minutes, i.e., after one hour the customer skips waiting and calls the vendor, asking for the pizza. We now assume that the clerk promises the pizza to be delivered soon, and the customers waits for the pizza again, asking again after the next 60 minutes, and so on. Let's have a closer look at the vendor process now. It is triggered by the order of the customer, as shown with the message start event and the message flow going from “order a pizza” to that event. After baking the pizza, the delivery boy will deliver the pizza and receive the payment, which includes giving a receipt to the customer.

In this example, we use message objects not only for informational objects, as the pizza order, but also for physical objects, like the pizza or the money. We can do this, because those physical objects actually act as informational objects inherently: When the pizza arrives at the customer's door, she will recognize this arrival and therefore know that the pizza has arrived, which is exactly the purpose of the accordant message event in the customer's pool. Of course, we can only use the model in that way because this example is not meant to be executed by a process engine.

5.3 Order Fulfillment and Procurement

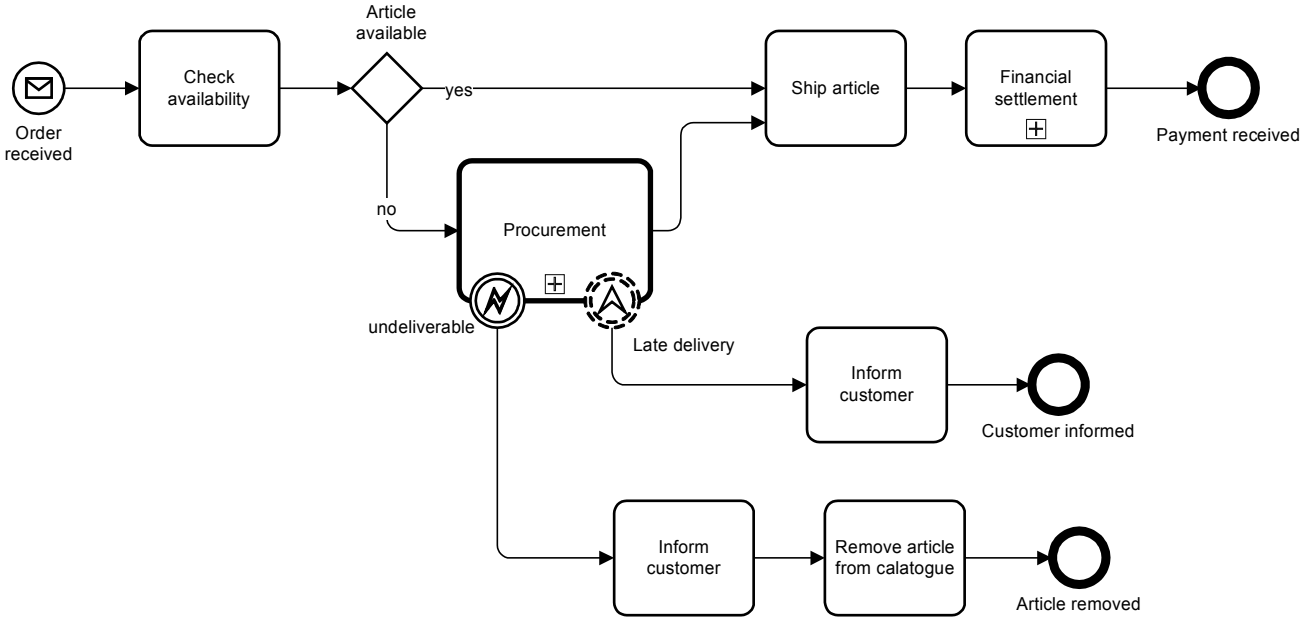


Figure 5.3: Order Fulfillment

This order fulfillment process starts after receiving an order message and continues to check whether the ordered article is available or not. An available article is shipped to the customer followed by a financial settlement, which is a collapsed sub-process in this diagram. In case that an article is not available, it has to be procured by calling the procurement sub-process. Please note that the shape of this collapsed sub-process is thickly bordered which means that it is a call activity. It is like a wrapper for a globally defined task or, like in this case, sub-process.

Another characteristic of the procurement sub-process are the two attached events. By using attached events it is possible to handle events that can spontaneously occur during the execution of a task or sub-process. Thereby we have to distinguish between interrupting and non-interrupting attached events. Both of them catch and handle the occurring events, but only the non-interrupting type (here it is the escalation event “late delivery”) does not abort the activity it is attached to. When the interrupting event type triggers, the execution of the current activity stops immediately.

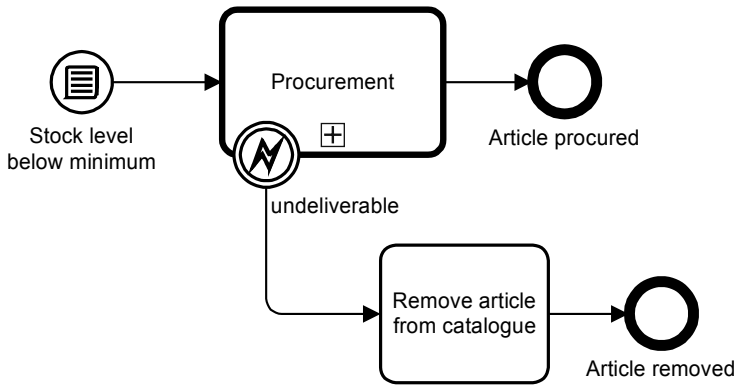


Figure 5.4: Stock maintenance process

The process for the stock maintenance is triggered by a conditional start event. It means that the process is instantiated in case that the condition became true, so in this example when the stock level goes below a certain minimum. In order to increase the stock level an article has to be procured. Therefore we use the same Procurement process as in the order fulfillment and refer to it by the call activity "Procurement", indicated by the thick border. Similar to the order fulfillment process this process handles the error exception by removing the article from the catalog. But in this stock maintenance process there appears to be no need for the handling of a "late delivery" escalation event. That's why it is left out and not handled. If the procurement sub-process finishes normally, the stock level is above minimum and the Stock Maintenance process ends with the end event “article procured”.

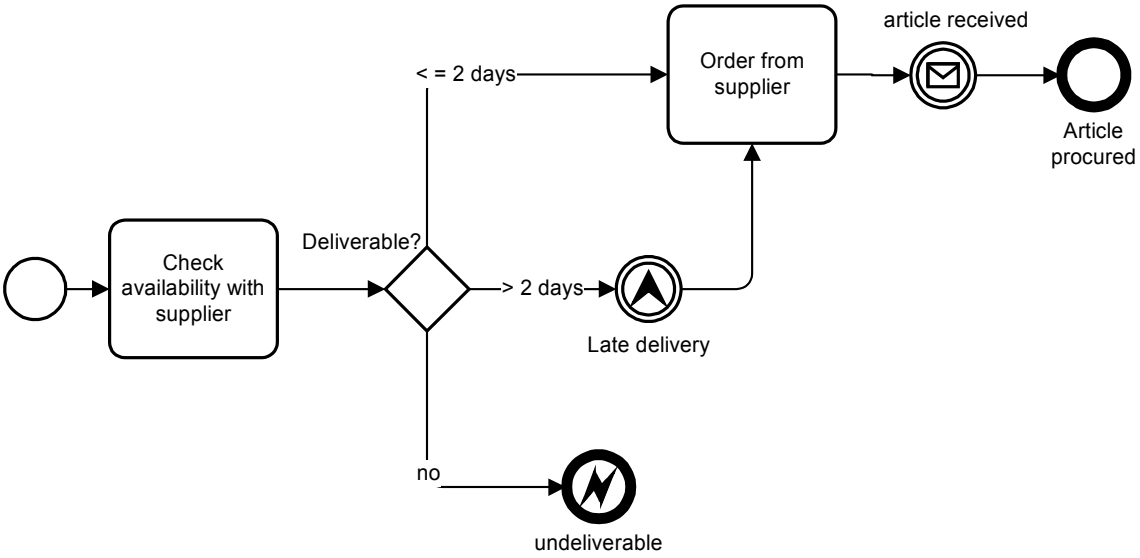


Figure 5.5: Procurement sub-process

We now zoom into the global sub-process “procurement” that is used by both order fulfillment and stock maintenance. Because this is a sub-process, the start event is plain, indicating that this process is not triggered by any external event but the referencing top-level-process.

The first task in this sub-process is the check whether the article to procured is available at the supplier. If not, this sub-process will throw the “not deliverable”-exception that is caught by both order fulfillment and stock maintenance, as we already discussed.

In case that the delivery in the Procurement process lasts more than 2 days an escalation event is thrown by the sub-process telling the referencing top-level-process that the delivery will be late. Similar to the error event, the escalation event has also an escalationCode which is necessary for the connection between throwing and catching escalation events. Contrary to the throwing error event, currently active threads are neither terminated nor affected by the throwing

intermediate escalation event. Furthermore, the Procurement process continues its execution by waiting for the delivery. But the thrown event is handled by the nearest parent activity with an attached intermediate escalation event which has the same escalationCode as the thrown escalation event. In the order fulfillment process, the "late delivery" escalation event attached to the Procurement sub-process catches the thrown "late delivery" event. But now, the event is a non-interrupting event. Because of that a new token is produced, follows the path of the escalation handling and triggers the task that informs the customer that the ordered article will be shipped later. When the procurement sub-process finishes, the Order Fulfillment process continues with the shipment of the article and the financial settlement.

6 Incident management

In this chapter we want to show the different perspectives you can take on the same business process, using BPMN. In the first step we will provide a rather simple, easy to read diagram that shows an incident process from a high level point of view. Later on we refine this model by moving from orchestration to collaboration and choreography. In the last step we take the organizational collaboration and imagine how a process engine could drive part of the process by user task assignments. The main purpose of this chapter is to demonstrate how you can use BPMN for creating simple and rather abstract diagrams, but also detailed views on human collaboration and finally for technical specifications for process execution.

6.1 High level model for quick understanding

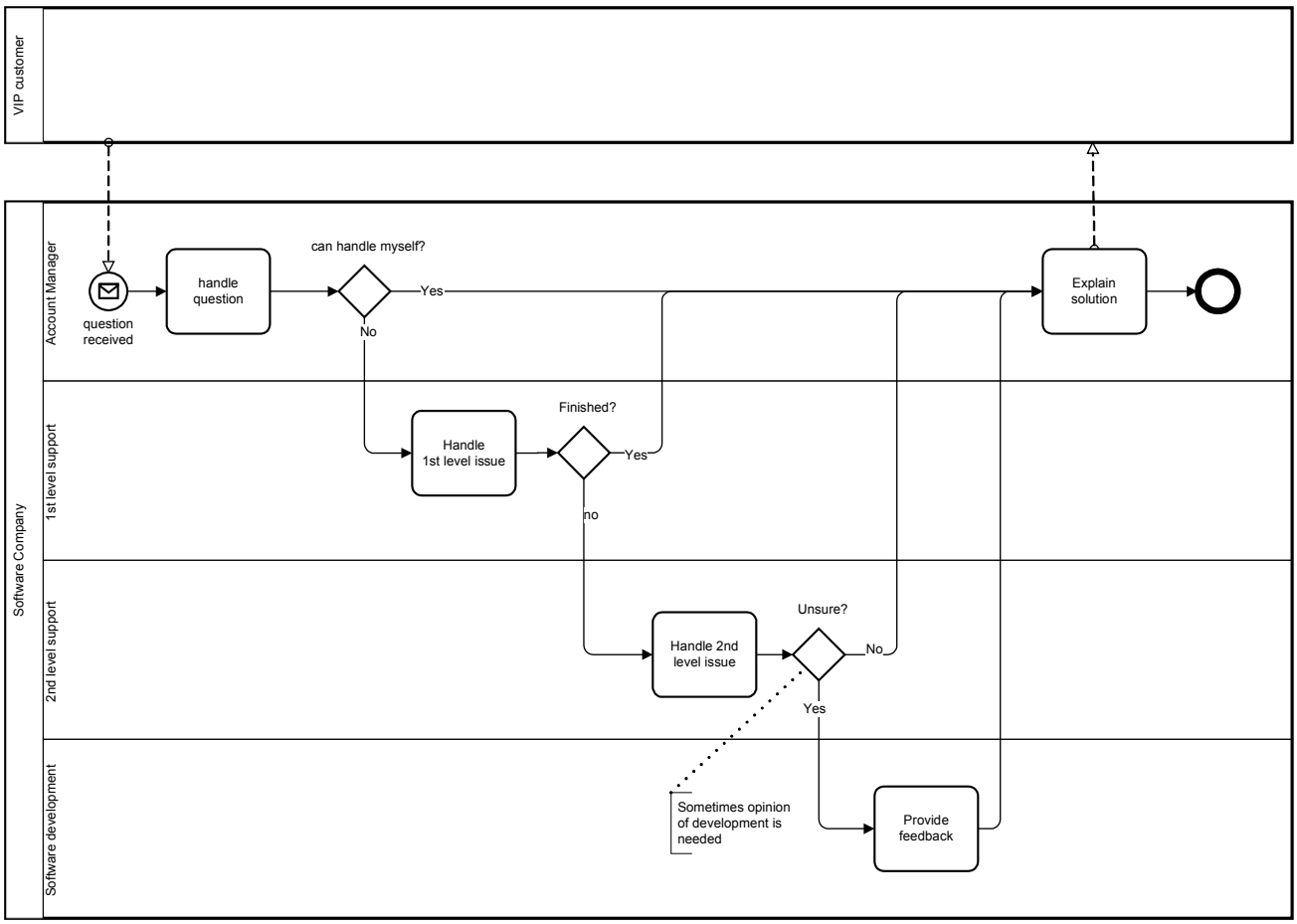


Figure 6.1: Incident management from high level point of view

The shown incident management process of a software manufacturer is triggered by a customer requesting help from her account manager because of a problem in the purchased product. First of all, the account manager should try to handle that request on his own and explain the solution to the customer, if possible. If not, the account manager will hand over the issue to a 1st level support agent, who will hand over to 2nd level support, if necessary. The 2nd level support agent should figure out if the customer can fix the problem on her own, but if the agent is not sure about this he can also ask a software developer for his opinion. In any case, at the end the account manager will explain the solution to the customer.

This diagram is really simple and somehow a "happy path", because we assume that we always find a solution we can finally explain to the customer. The model lacks all details of collaboration between the involved employees, and the abstract tasks indicate that we do not have any information about whether the process or parts of it are executable by a

process engine. This diagram is useful, if you want to scope the process, get a basic understanding of the flow, and talk about the main steps, but not if you want to dig into the details for discussing process improvements or even software driven support of the process.

6.2 Detailed Collaboration and Choreography

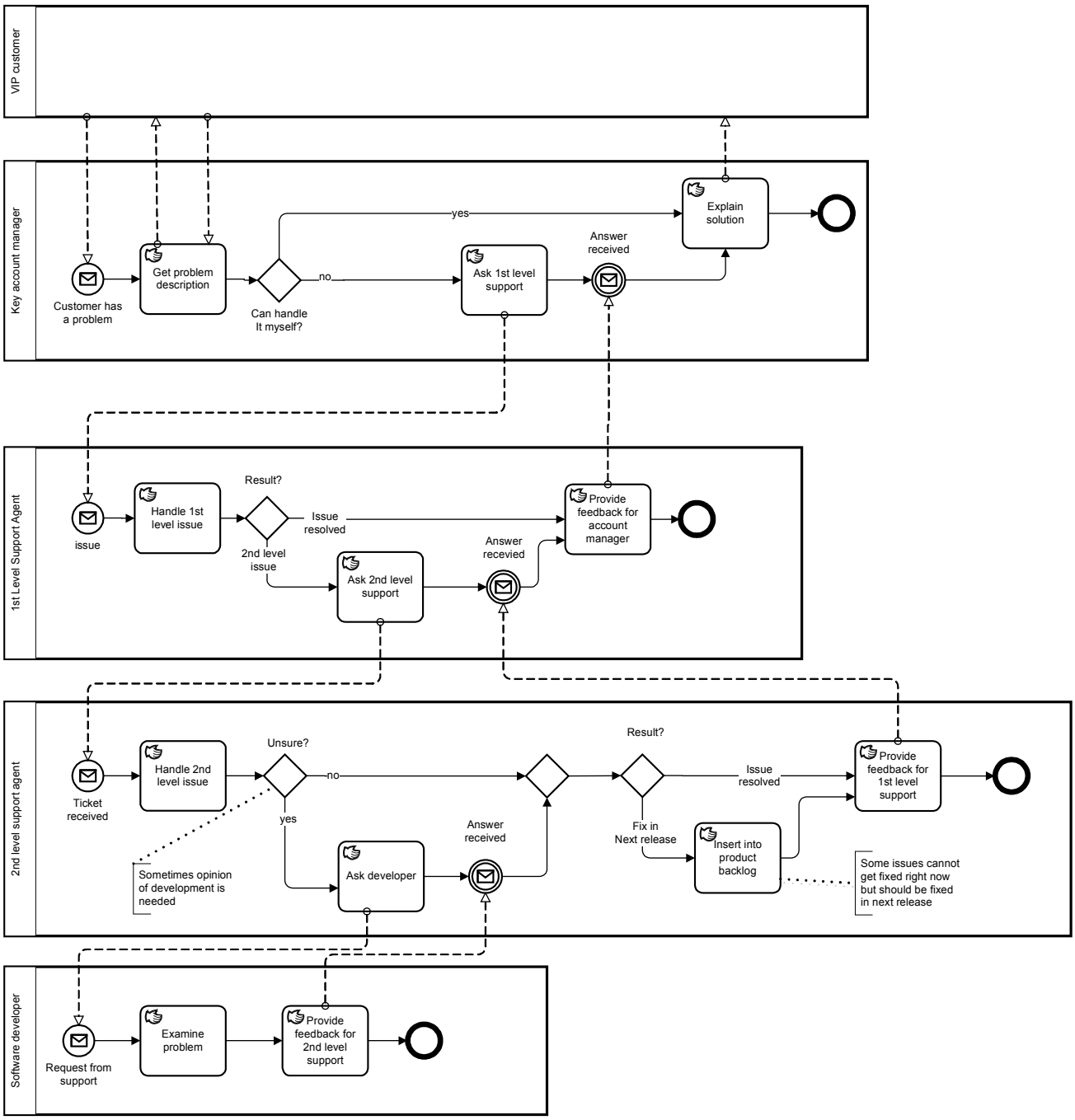


Figure 6.2: Incident Management as detailed collaboration

We can take a closer look at the ping-pong-game of account manager, support agents and software developer by switching from a single-pool-model to a collaboration diagram, as shown above. We can now see some more details about the particular processes each participant fulfills, e.g., the dialogue between the account manager and the customer for clarifying the customer's problem, or the fact that the 2nd level support agent will insert a request for a feature in the

product backlog, if the current release of the software product cannot cover the customer's demand satisfactorily. We have also specified each task as manual, which means that we still think of the processes as completely human-driven with no process engine involved. This could hypothetically be the As-Is-state of the incident management before the introduction of a process engine. The next step could be to define whether we want to drive the complete collaboration by a process engine, or only parts of it. But before we discuss that matter, we can have a look at an other way of modeling such a ping-pong-game, the choreography diagram shown below. This diagram only shows the tasks that are dedicated to the communication between the different process participants, hiding all internal steps, e.g., the task that inserts a new entry into the product backlog. Note that the diagrams shown in Figure 6.1 and 6.2 have no formal connection between each other, whereas the Figure 6.2 and 6.3 have the exact same semantic model behind them and just provide different views on it. See also Annex A for an XML serialization of the underlying semantic model.

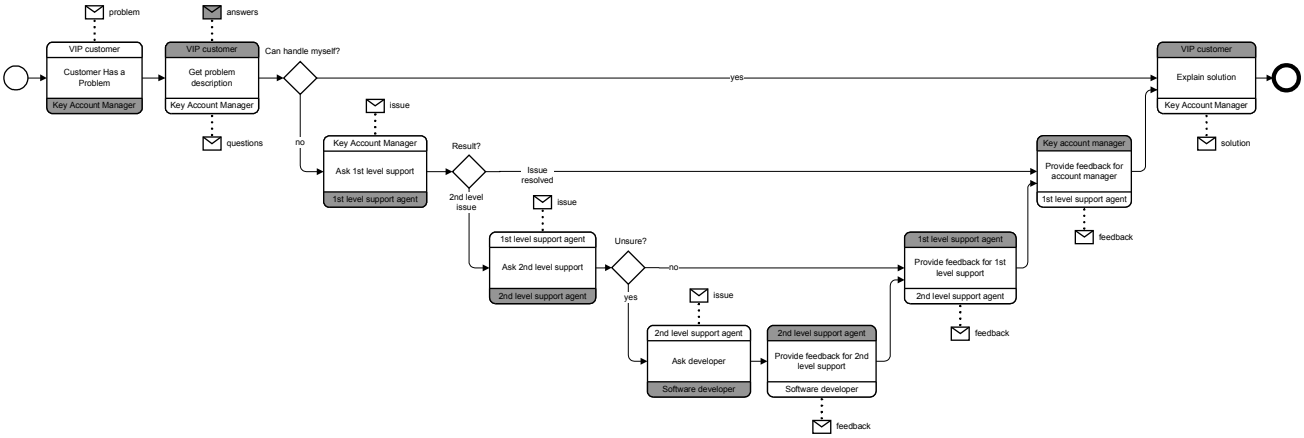


Figure 6.3: Incident Management as choreography

6.3 Human-driven vs. system-driven control flows

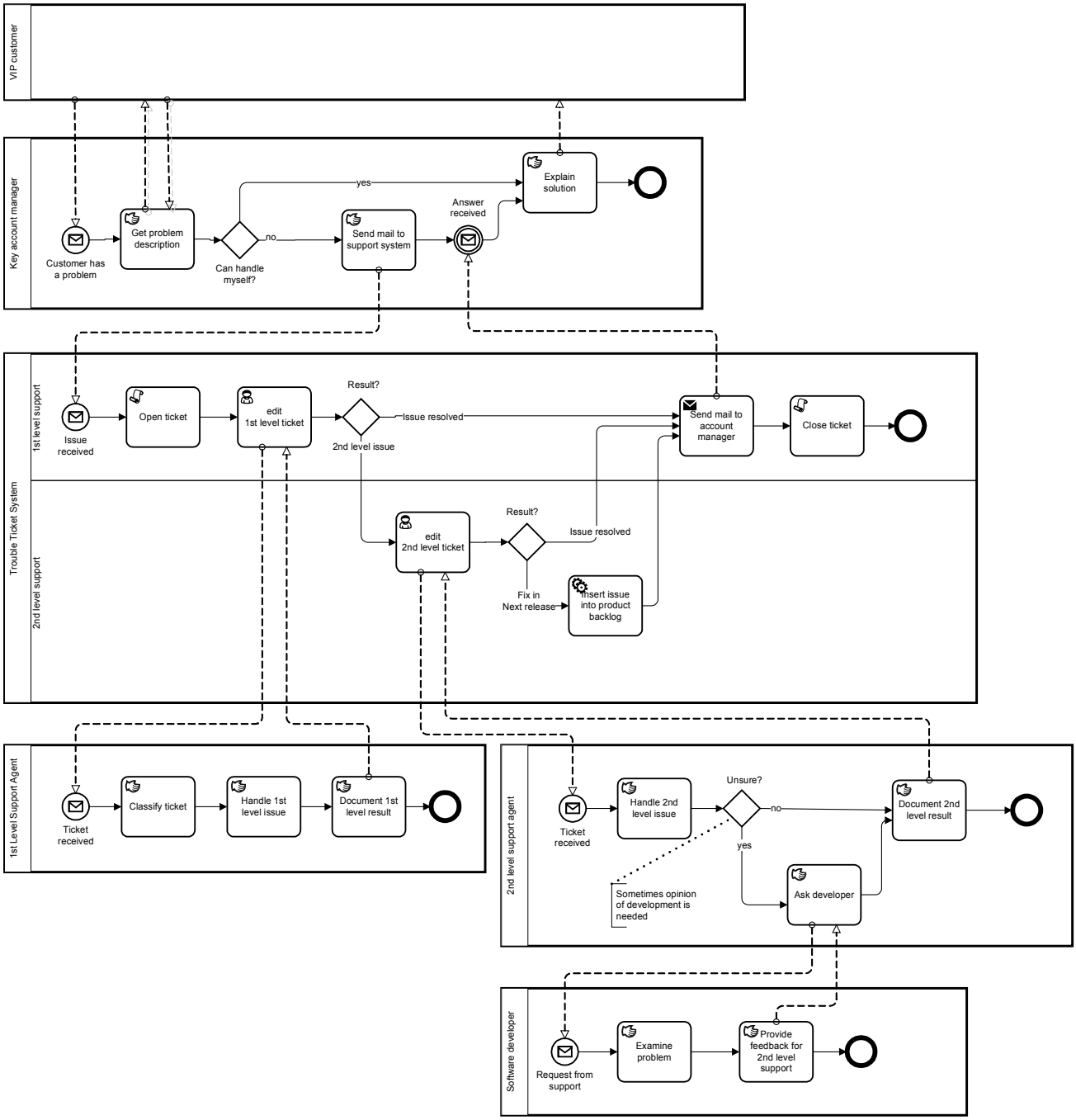


Figure 6.4: Incident Management with human-driven and system-driven pools

If we imagine we are realizing a project for automating the incident management process, we could now decide which parts of it should be actually executed in a process engine, and which parts should remain human-driven. In this scenario we decided that the account manager should not be bothered with web forms or task lists, he should just send an email if he wants to report a customer's problem, and receive an email when the process has completed. The same idea applies for the software developer: Let us assume the 2nd level support agent sits in the same room as the developers. Maybe it is more efficient if the support agent just walks over to the developer and talks about the issue, rather than playing some time consuming ping-pong-game with task assignments. Therefore, we want to keep this part of the incident management human driven as well: no process engine driving the collaboration between 2nd level support and software developers. But

we do want the assignment of tickets to 1st and 2nd level support agents by a trouble ticket system, which now takes the role of the process engine and therefore is modeled in a dedicated pool. That system can actually receive and parse emails sent by the account manager and opens a ticket for it. If the 1st level support agent decides that this is a 2nd level issue, he does so by documenting his decision and completing the assigned task "edit 1st level ticket". The trouble ticket system then routes the ticket to the 2nd level support agent. When that agent has finished, he maybe declared the issue to be fixed in the next software release. Then the trouble ticket system makes a service call on the product backlog system, a new feature we have introduced with our process engine: The entry does not have to be inserted manually any more. In the end, the trouble ticket system will send an email to the account manager, containing the results of the incident management, and close the ticket. The account manager can then explain the solution to the customer based on the information in the ticket.

Of course, this way of modeling both human-driven and system-driven control flows in one diagram is just a proposal, that should give an idea of useful modeling approaches based on collaboration diagrams. It should demonstrate how BPMN could support Business-IT-Alignment in process modeling: We can hand over the modeled process engine pool to an actual process engine for execution, while we can show the other pools separately to our process participants, the support agents or the account manager, and discuss their involvement in the collaboration based on those simplified views on the same, consistent collaboration model. This gives us the opportunity to talk with both Business people and IT people about the same process model, without overburdening business people with too complex diagrams or IT people with too inaccurate process models.

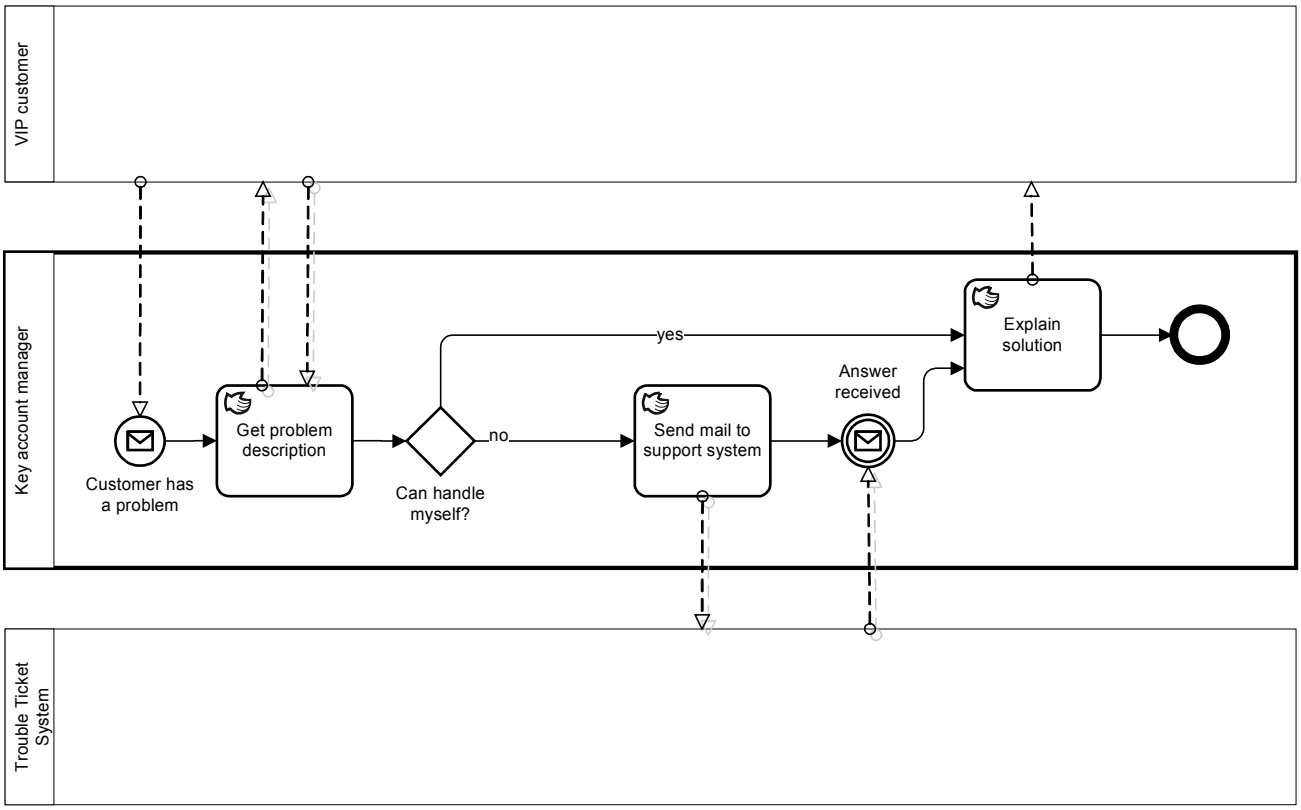


Figure 6.5: This rather simple diagram is all we have to show to the account manager

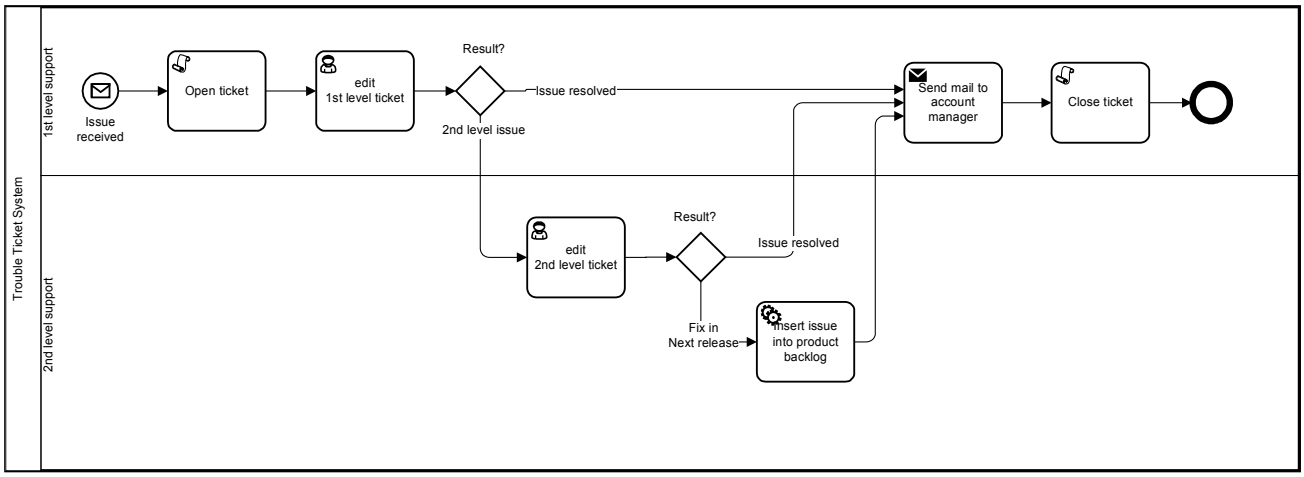


Figure 6.6: This is the only part of the whole collaboration we will execute in a process engine

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<semantic:definitions id="_1275940773761" targetNamespace="http://www.trisotech.com/definitions/_1275940773761"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
 xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
 xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
 xmlns:semantic="http://www.omg.org/spec/BPMN/20100524/MODEL">
   <semantic:message id="_1275940773779"/>
   <semantic:message id="_1275940773809"/>
   <semantic:process isExecutable="false" id="_1-1">
     <semantic:laneSet id="ls_1-1">
       <semantic:lane name="1st level support" id="_1-9">
       <semantic:lane name="2nd level support" id="_1-11">
     </semantic:laneSet>
     <semantic:startEvent name="Issue received" id="_1-13">
     <semantic:scriptTask completionQuantity="1" isForCompensation="false" startQuantity="1" name="Open ticket" id="_1-26">
     <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="edit  1st level ticket" id="_1-77">
     <semantic:exclusiveGateway gatewayDirection="Unspecified" name="Result?" id="_1-128">
     <semantic:sendTask implementation="##WebService" messageRef="_1275940773809" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Send mail to account manager" id="_1-150">
     <semantic:scriptTask completionQuantity="1" isForCompensation="false" startQuantity="1" name="Close ticket" id="_1-201">
     <semantic:endEvent name="" id="_1-376">
     <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="edit  2nd level ticket" id="_1-252">
     <semantic:exclusiveGateway gatewayDirection="Unspecified" name="Result?" id="_1-303">
     <semantic:serviceTask implementation="##WebService" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Insert issue into product backlog" id="_1-325">
     <semantic:sequenceFlow sourceRef="_1-13" targetRef="_1-26" name="" id="_1-390"/>
     <semantic:sequenceFlow sourceRef="_1-26" targetRef="_1-77" name="" id="_1-392"/>
     <semantic:sequenceFlow sourceRef="_1-77" targetRef="_1-128" name="" id="_1-394"/>
     <semantic:sequenceFlow sourceRef="_1-128" targetRef="_1-150" name="Issue resolved" id="_1-396"/>
     <semantic:sequenceFlow sourceRef="_1-150" targetRef="_1-201" name="" id="_1-398"/>
     <semantic:sequenceFlow sourceRef="_1-201" targetRef="_1-376" name="" id="_1-400"/>
     <semantic:sequenceFlow sourceRef="_1-128" targetRef="_1-252" name="2nd level issue" id="_1-402"/>
     <semantic:sequenceFlow sourceRef="_1-252" targetRef="_1-303" name="" id="_1-404"/>
     <semantic:sequenceFlow sourceRef="_1-303" targetRef="_1-150" name="Issue resolved" id="_1-406"/>
     <semantic:sequenceFlow sourceRef="_1-325" targetRef="_1-150" name="" id="_1-408"/>
     <semantic:sequenceFlow sourceRef="_1-303" targetRef="_1-325" name="Fix in Next release" id="_1-410"/>
   </semantic:process>
   <semantic:collaboration id="C1275940773964">
     <semantic:participant name="Trouble Ticket System" processRef="_1-1" id="_1-1"/>
   </semantic:collaboration>
   <bpmndi:BPMNDiagram documentation="" id="Trisotech.Visio-_1" name="Process Engine Only" resolution="96.00000267028808">
</semantic:definitions>
```

Figure 6.7: XML serialization for process engine pool.

Table 6.1: Process engine pool enriched with execution details. This is what a process engine would execute.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<definitions name="Incident Management" id="_98a0678d9e194de9b3d9284886c3"
    targetNamespace="http://fox.camunda.com/model/98a0678d9e194de9b3d9284886c3"
    xmlns:tns="http://fox.camunda.com/model/98a0678d9e194de9b3d9284886c3"
    xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
    xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
    xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
    xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:java="http://jcp.org/en/jsr/detail?id=270"
    typeLanguage="http://jcp.org/en/jsr/detail?id=270"
    expressionLanguage="http://www.jcp.org/en/jsr/detail?id=245">
    <!--
        Java SE 6 is used as type language for the model whereas the Java
        Unified Expression Language serves as language for Expressions.
    -->

    <collaboration id="C1275940773964">
        <participant name="Trouble Ticket System" processRef="tns:WFP-1-1"
            id="_1-1" />
    </collaboration>

    <process isExecutable="true" id="WFP-1-1">

        <ioSpecification>
            <dataInput itemSubjectRef="tns:IssueItem" id="IssueDataInputOfProcess" />
            <inputSet>
                <dataInputRefs>IssueDataInputOfProcess</dataInputRefs>
            </inputSet>
            <outputSet></outputSet>
        </ioSpecification>

        <!--
            This Lane Set partitions the Flow Nodes of the Process according to
            the Resources that are responsible for them. However, this does not
            affect the actual assignment of Resources to Activities as meaning
            of the Lanes is up to the modeler and not specified in BPMN.
        -->
        <laneSet id="ls_1-1">
            <lane name="1st level support"
                partitionElementRef="tns:FirstLevelSupportResource" id="_1-9">
                <flowNodeRef>_1-13</flowNodeRef>
                <flowNodeRef>_1-26</flowNodeRef>
                <flowNodeRef>_1-77</flowNodeRef>
                <flowNodeRef>_1-128</flowNodeRef>
                <flowNodeRef>_1-150</flowNodeRef>
                <flowNodeRef>_1-201</flowNodeRef>
                <flowNodeRef>_1-376</flowNodeRef>
            </lane>
            <lane name="2nd level support"
                partitionElementRef="tns:SecondLevelSupportResource" id="_1-11">
                <flowNodeRef>_1-252</flowNodeRef>
                <flowNodeRef>_1-303</flowNodeRef>
                <flowNodeRef>_1-325</flowNodeRef>
            </lane>
        </laneSet>

        <startEvent name="Issue received" id="_1-13">
            <dataOutput itemSubjectRef="tns:IssueItem"
                id="IssueDataOutputOfStartEvent" />
            <dataOutputAssociation>
                <sourceRef>IssueDataOutputOfStartEvent</sourceRef>
                <targetRef>IssueDataInputOfProcess</targetRef>
            </dataOutputAssociation>
            <messageEventDefinition messageRef="tns:IssueMessage" />
```

```
</startEvent>

<sequenceFlow sourceRef="_1-13" targetRef="_1-26" id="_1-390" />

<!--
    This script task uses the Groovy programming language to create a
    Data Object and fill it with data of the Item received in the
    Message that started the Process.
-->
<scriptTask name="Open ticket" scriptFormat="text/x-groovy" id="_1-26">
    <ioSpecification>
        <dataInput itemSubjectRef="tns:IssueItem"
            id="IssueDataInputOfScriptTask" />
        <dataOutput itemSubjectRef="tns:TicketItem" id="TicketDataOutputOfScriptTask"/>
        <inputSet>
            <dataInputRefs>IssueDataInputOfScriptTask</dataInputRefs>
        </inputSet>
        <outputSet>
            <dataOutputRefs>TicketDataOutputOfScriptTask</dataOutputRefs>
        </outputSet>
    </ioSpecification>
    <dataInputAssociation>
        <sourceRef>IssueDataInputOfProcess</sourceRef>
        <targetRef>IssueDataInputOfScriptTask</targetRef>
    </dataInputAssociation>
    <dataOutputAssociation>
        <sourceRef>TicketDataOutputOfScriptTask</sourceRef>
        <targetRef>TicketDataObject</targetRef>
    </dataOutputAssociation>
    <script><![CDATA[
        issueReport = getDataInput("IssueDataInputOfScriptTask")

        ticket = new TroubleTicket()
        ticket.setDate = new Date()
        ticket.setState = "Open"
        ticket.setReporter = issueReport.getAuthor()
        ticket.setDesctiption = issueReport.getText()

        setDataOutput("TicketDataOutputOfScriptTask", ticket)
    ]]></script>
</scriptTask>

<dataObject id="TicketDataObject" itemSubjectRef="tns:TicketItem" />

<sequenceFlow sourceRef="_1-26" targetRef="_1-77" id="_1-392" />

<userTask name="edit 1st level ticket" id="_1-77">
    <ioSpecification>
        <dataInput itemSubjectRef="tns:TicketItem" id="TicketDataInputOf_1-77" />
        <dataOutput itemSubjectRef="tns:TicketItem" id="TicketDataOutputOf_1-77" />
        <inputSet>
            <dataInputRefs>TicketDataInputOf_1-77</dataInputRefs>
        </inputSet>
        <outputSet>
            <dataOutputRefs>TicketDataOutputOf_1-77</dataOutputRefs>
        </outputSet>
    </ioSpecification>
    <dataInputAssociation>
        <sourceRef>TicketDataObject</sourceRef>
        <targetRef>TicketDataInputOf_1-77</targetRef>
    </dataInputAssociation>
    <dataOutputAssociation>
        <sourceRef>TicketDataOutputOf_1-77</sourceRef>
        <targetRef>TicketDataObject</targetRef>
    </dataOutputAssociation>
    <potentialOwner>
        <resourceRef>tns:FirstLevelSupportResource</resourceRef>
```

```
    </potentialOwner>
</userTask>

<sequenceFlow sourceRef="_1-77" targetRef="_1-128" id="_1-394" />

<exclusiveGateway name="Result?" gatewayDirection="Diverging"
    id="_1-128" />

<sequenceFlow sourceRef="_1-128" targetRef="_1-252"
    name="2nd level issue" id="_1-402">
    <conditionExpression xsi:type="tFormalExpression">
        ${getDataObject("TicketDataObject").status == "Open"}
    </conditionExpression>
</sequenceFlow>
<sequenceFlow sourceRef="_1-128" targetRef="_1-150"
    name="Issue resolved" id="_1-396">
    <conditionExpression xsi:type="tFormalExpression">
        ${getDataObject("TicketDataObject").status == "Resolved"}
    </conditionExpression>
</sequenceFlow>

<userTask name="edit 2nd level ticket" id="_1-252">
    <ioSpecification>
        <dataInput itemSubjectRef="tns:TicketItem" id="TicketDataInputOf_1-252" />
        <dataOutput itemSubjectRef="tns:TicketItem" id="TicketDataOutputOf_1-252" />
        <inputSet>
            <dataInputRefs>TicketDataInputOf_1-252</dataInputRefs>
        </inputSet>
        <outputSet>
            <dataOutputRefs>TicketDataOutputOf_1-252</dataOutputRefs>
        </outputSet>
    </ioSpecification>
    <dataInputAssociation>
        <sourceRef>TicketDataObject</sourceRef>
        <targetRef>TicketDataInputOf_1-252</targetRef>
    </dataInputAssociation>
    <dataOutputAssociation>
        <sourceRef>TicketDataOutputOf_1-252</sourceRef>
        <targetRef>TicketDataObject</targetRef>
    </dataOutputAssociation>
    <potentialOwner>
        <resourceRef>tns:SecondLevelSupportResource</resourceRef>
    </potentialOwner>
</userTask>

<sequenceFlow sourceRef="_1-252" targetRef="_1-303" id="_1-404" />

<exclusiveGateway name="Result?" gatewayDirection="Diverging"
    id="_1-303" />

<sequenceFlow sourceRef="_1-303" targetRef="_1-325"
    name="Fix in Next release" id="_1-410">
    <conditionExpression xsi:type="tFormalExpression">
        ${getDataObject("TicketDataObject").status == "Deferred"}
    </conditionExpression>
</sequenceFlow>

<sequenceFlow sourceRef="_1-303" targetRef="_1-150"
    name="Issue resolved" id="_1-406">
    <conditionExpression xsi:type="tFormalExpression">
        ${getDataObject("TicketDataObject").status == "Resolved"}
    </conditionExpression>
</sequenceFlow>

<serviceTask name="Insert issue into product backlog"
    operationRef="tns:addTicketOperation" id="_1-325">
    <ioSpecification>
```

```
            <dataInput itemSubjectRef="tns:TicketItem" id="TicketDataInputOf_1-325" />
            <inputSet>
                <dataInputRefs>TicketDataInputOf_1-325</dataInputRefs>
            </inputSet>
            <outputSet />
        </ioSpecification>
        <dataInputAssociation>
            <sourceRef>TicketDataObject</sourceRef>
            <targetRef>TicketDataInputOf_1-325</targetRef>
        </dataInputAssociation>
    </serviceTask>

    <sequenceFlow sourceRef="_1-325" targetRef="_1-150" id="_1-408" />

    <sendTask name="Send mail to account manager" messageRef="tns:AnswerMessage"
        operationRef="tns:sendMailToIssueReporterOperation" id="_1-150">
        <ioSpecification>
            <dataInput itemSubjectRef="tns:AnswerItem" id="AnswerDataInputOfSendTask" />
            <inputSet>
                <dataInputRefs>AnswerDataInputOfSendTask</dataInputRefs>
            </inputSet>
            <outputSet />
        </ioSpecification>
        <dataInputAssociation>
            <sourceRef>TicketDataObject</sourceRef>
            <targetRef>AnswerDataInputOfSendTask</targetRef>
            <assignment>
                <from>${getDataObject("TicketDataObject").reporter}</from>
                <to>${getDataInput("AnswerDataInputOfSendTask").recipient</to>
            </assignment>
            <assignment>
                <from>
                    A ticket has been created for your issue, which is now in
                    status ${getDataObject("TicketDataObject").status}.
                </from>
                <to>${getDataInput("AnswerDataInputOfSendTask").body}</to>
            </assignment>
        </dataInputAssociation>
    </sendTask>

    <sequenceFlow sourceRef="_1-150" targetRef="_1-201" id="_1-398" />

    <scriptTask name="Close ticket" scriptFormat="text/x-groovy"
        id="_1-201">
        <ioSpecification>
            <dataInput itemSubjectRef="tns:TicketItem" id="TicketDataInputOf_1-398" />
            <inputSet>
                <dataInputRefs>TicketDataInputOf_1-398</dataInputRefs>
            </inputSet>
            <outputSet />
        </ioSpecification>
        <dataInputAssociation>
            <sourceRef>TicketDataObject</sourceRef>
            <targetRef>TicketDataInputOf_1-398</targetRef>
        </dataInputAssociation>
        <script><![CDATA[
            ticket = getDataInput("TicketDataInputOf_1-398")
            ticket.close()
        ]]></script>
    </scriptTask>

    <sequenceFlow sourceRef="_1-201" targetRef="_1-376" id="_1-400" />

    <endEvent id="_1-376" />

</process>
```

```
    <resource id="FirstLevelSupportResource" name="1st Level Support" />

    <resource id="SecondLevelSupportResource" name="2nd Level Support" />

    <interface name="Product Backlog Interface"
        implementationRef="java:com.camunda.examples.incidentmanagement.ProductBacklog">
        <operation name="addTicketOperation" implementationRef="addTicket"
            id="addTicketOperation">
            <inMessageRef>tns:AddTicketMessage</inMessageRef>
        </operation>
    </interface>

    <interface name="Mail Interface"
        implementationRef="java:com.camunda.examples.incidentmanagement.Mail">
        <operation name="sendMailToIssueReporterOperation" implementationRef="sendMail"
            id="sendMailToIssueReporterOperation">
            <inMessageRef>tns:AnswerMessage</inMessageRef>
        </operation>
    </interface>

    <message id="IssueMessage" name="Issue Message" itemRef="tns:IssueItem" />

    <message id="AddTicketMessage" name="addTicket Message"
        itemRef="tns:TicketItem" />

    <message id="AnswerMessage" name="Answer Message" itemRef="tns:AnswerItem" />

    <itemDefinition id="IssueItem" isCollection="false" itemKind="Information"
        structureRef="com.camunda.examples.incidentmanagement.IssueReport" />

    <itemDefinition id="TicketItem" isCollection="false" itemKind="Information"
        structureRef="com.camunda.examples.incidentmanagement.TroubleTicket" />

    <itemDefinition id="AnswerItem" isCollection="false" itemKind="Information"
        structureRef="com.camunda.examples.incidentmanagement.Answer" />

</definitions>
```

7 Models and Diagrams

The purpose of this chapter is to demonstrate via examples some of the interrelations between models and diagrams. We explore how different BPMN diagrams of the same scenario lead to different serializations of the model.

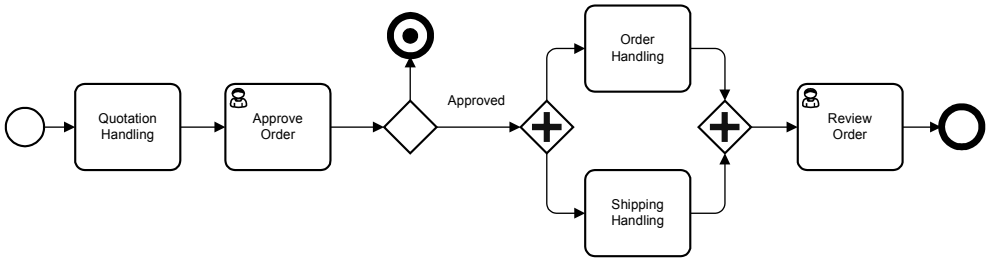
The process scenario used in the examples from this chapter is inspired from figure 10.24 of the BPMN 2.0 Specification document.

7.1 Lane and Pool

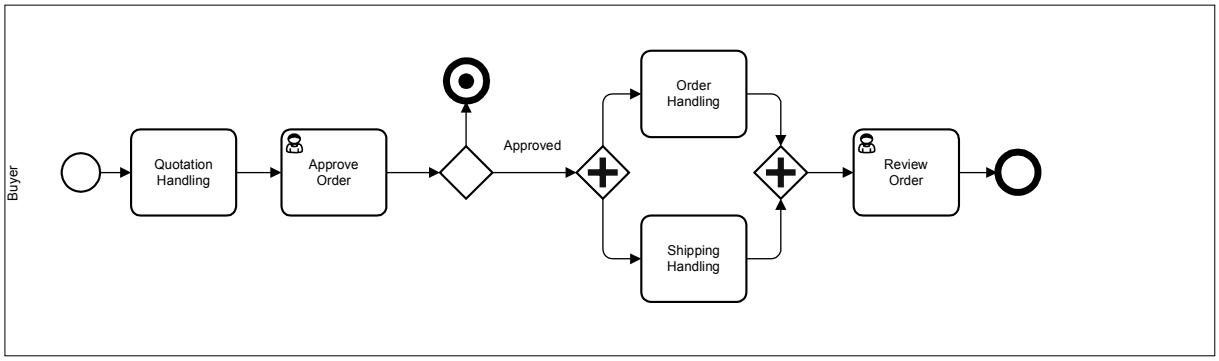
In this section, we explore the use of lanes and pools in a BPMN diagram and their corresponding serializations.

7.1.1 Lane

A process can be depicted in a Process Diagram with or without lanes. Both these depictions lead to one process in the model and one diagram of that process. The main difference in the two serializations is that one does not have a Laneset with a lane in it, while the other does.



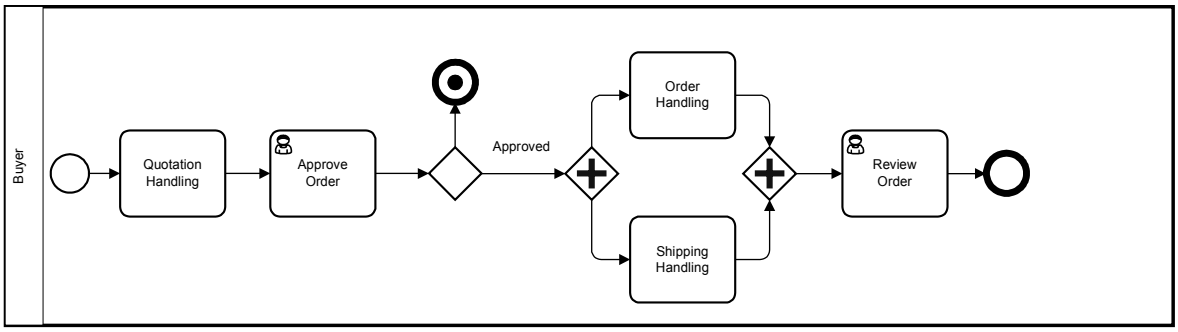

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<semantic:definitions id="_1275486223307" targetNamespace="http://www.trisotech.com/definitions/_1275486223307"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
xmlns:semantic="http://www.omg.org/spec/BPMN/20100524/MODEL">
  <semantic:process isExecutable="false" id="_6">
    <semantic:startEvent name="" id="StartProcess">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Quotation Handling" id="TaskQuotationHandling">
    <semantic:exclusiveGateway gatewayDirection="Diverging" name="" id="GatewayOrderApprovedDecision">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Order  Handling" id="_6-190">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Shipping Handling" id="_6-241">
    <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Review Order" id="TaskReviewOrder">
    <semantic:endEvent name="" id="EndProcess">
    <semantic:parallelGateway gatewayDirection="Diverging" name="" id="ParaSplitOrderAndShipment">
    <semantic:parallelGateway gatewayDirection="Converging" name="" id="ParaJoinOderAndShipment">
    <semantic:endEvent name="" id="TerminateProcess">
    <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Approve Order" id="TaskApproveOrder">
    <semantic:sequenceFlow sourceRef="StartProcess" targetRef="TaskQuotationHandling" name="" id="_6-468"/>
    <semantic:sequenceFlow sourceRef="TaskQuotationHandling" targetRef="TaskApproveOrder" name="" id="_6-470"/>
    <semantic:sequenceFlow sourceRef="TaskApproveOrder" targetRef="GatewayOrderApprovedDecision" name="" id="_6-500"/>
    <semantic:sequenceFlow sourceRef="GatewayOrderApprovedDecision" targetRef="ParaSplitOrderAndShipment" name="Approved" id="_6-502"/>
    <semantic:sequenceFlow sourceRef="ParaSplitOrderAndShipment" targetRef="_6-190" name="" id="_6-504"/>
    <semantic:sequenceFlow sourceRef="ParaSplitOrderAndShipment" targetRef="_6-241" name="" id="_6-506"/>
    <semantic:sequenceFlow sourceRef="_6-190" targetRef="ParaJoinOderAndShipment" name="" id="_6-508"/>
    <semantic:sequenceFlow sourceRef="_6-241" targetRef="ParaJoinOderAndShipment" name="" id="_6-532"/>
    <semantic:sequenceFlow sourceRef="ParaJoinOderAndShipment" targetRef="TaskReviewOrder" name="" id="_6-534"/>
    <semantic:sequenceFlow sourceRef="TaskReviewOrder" targetRef="EndProcess" name="" id="_6-536"/>
    <semantic:sequenceFlow sourceRef="GatewayOrderApprovedDecision" targetRef="TerminateProcess" name="" id="_6-552"/>
  </semantic:process>
  <bpmndi:BPMNDiagram documentation="" id="Trisotech.Visio-_6" name="Order Process" resolution="96.00000267028808">
    <bpmndi:BPMNPlane bpmnElement="_6">
  </bpmndi:BPMNDiagram>
</semantic:definitions>
```




```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<semantic:definitions id="_1275486169167" targetNamespace="http://www.trisotech.com/definitions/_1275486169167"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
xmlns:semantic="http://www.omg.org/spec/BPMN/20100524/MODEL">
  <semantic:process isExecutable="false" id="_6-1">
    <semantic:laneSet id="ls_1275486169372">
      <semantic:lane name="Buyer" id="Buyer">
    </semantic:laneSet>
    <semantic:startEvent name="" id="StartProcess">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Quotation Handling" id="TaskQuotationHandling">
    <semantic:exclusiveGateway gatewayDirection="Diverging" name="" id="GatewayOrderApprovedDecision">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Order  Handling" id="_6-190">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Shipping Handling" id="_6-241">
    <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Review Order" id="TaskReviewOrder">
    <semantic:endEvent name="" id="EndProcess">
    <semantic:parallelGateway gatewayDirection="Diverging" name="" id="ParaSplitOrderAndShipment">
    <semantic:parallelGateway gatewayDirection="Converging" name="" id="ParaJoinOderAndShipment">
    <semantic:endEvent name="" id="TerminateProcess">
    <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Approve Order" id="TaskApproveOrder">
    <semantic:sequenceFlow sourceRef="StartProcess" targetRef="TaskQuotationHandling" name="" id="_6-468"/>
    <semantic:sequenceFlow sourceRef="TaskQuotationHandling" targetRef="TaskApproveOrder" name="" id="_6-470"/>
    <semantic:sequenceFlow sourceRef="TaskApproveOrder" targetRef="GatewayOrderApprovedDecision" name="" id="_6-500"/>
    <semantic:sequenceFlow sourceRef="GatewayOrderApprovedDecision" targetRef="ParaSplitOrderAndShipment" name="Approved" id="_6-502"/>
    <semantic:sequenceFlow sourceRef="ParaSplitOrderAndShipment" targetRef="_6-190" name="" id="_6-504"/>
    <semantic:sequenceFlow sourceRef="ParaSplitOrderAndShipment" targetRef="_6-241" name="" id="_6-506"/>
    <semantic:sequenceFlow sourceRef="_6-190" targetRef="ParaJoinOderAndShipment" name="" id="_6-508"/>
    <semantic:sequenceFlow sourceRef="_6-241" targetRef="ParaJoinOderAndShipment" name="" id="_6-532"/>
    <semantic:sequenceFlow sourceRef="ParaJoinOderAndShipment" targetRef="TaskReviewOrder" name="" id="_6-534"/>
    <semantic:sequenceFlow sourceRef="TaskReviewOrder" targetRef="EndProcess" name="" id="_6-536"/>
    <semantic:sequenceFlow sourceRef="GatewayOrderApprovedDecision" targetRef="TerminateProcess" name="" id="_6-552"/>
  </semantic:process>
  <bpmndi:BPMNDiagram documentation="" id="Trisotech.Visio-_6" name="Order Process" resolution="96.00000267028808">
    <bpmndi:BPMNPlane bpmnElement="_6-1">
  </bpmndi:BPMNDiagram>
</semantic:definitions>
```

7.1.2 Pool

Pools are only present in Collaboration Diagrams (Collaborations, Choreographies, Conversations). Thus, when depicting the same scenario using a pool, we are producing a Collaboration Diagram. The introduction of a pool in our depiction implies that we are producing a Collaboration Diagram. In fact, this is a diagram of an incomplete Collaboration, as a Collaboration should be between two or more participants.



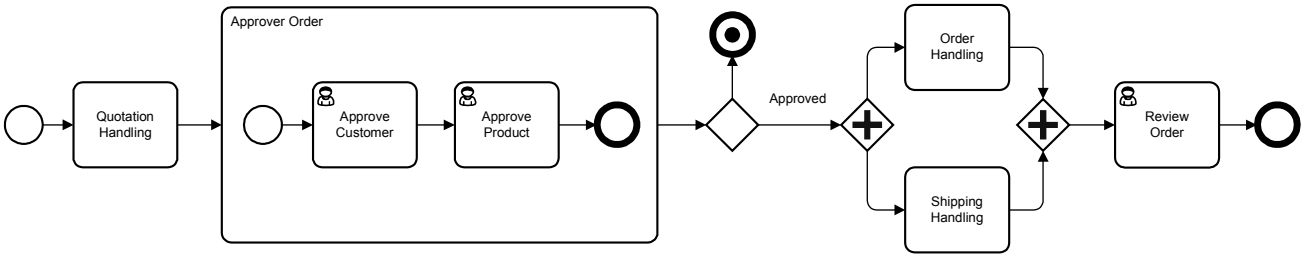

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<semantic:definitions id="_1275486197916" targetNamespace="http://www.trisotech.com/definitions/_1275486197916"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
 xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
 xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
 xmlns:semantic="http://www.omg.org/spec/BPMN/20100524/MODEL">
  <semantic:process isExecutable="false" id="_6-1">
    <semantic:startEvent name="" id="StartProcess">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Quotation Handling" id="TaskQuotationHandling">
    <semantic:exclusiveGateway gatewayDirection="Diverging" name="" id="GatewayOrderApprovedDecision">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Order  Handling" id="_6-190">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Shipping Handling" id="_6-241">
    <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Review Order" id="TaskReviewOrder">
    <semantic:endEvent name="" id="EndProcess">
    <semantic:parallelGateway gatewayDirection="Diverging" name="" id="ParaSplitOrderAndShipment">
    <semantic:parallelGateway gatewayDirection="Converging" name="" id="ParaJoinOderAndShipment">
    <semantic:endEvent name="" id="TerminateProcess">
    <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Approve Order" id="TaskApproveOrder">
    <semantic:sequenceFlow sourceRef="StartProcess" targetRef="TaskQuotationHandling" name="" id="_6-468"/>
    <semantic:sequenceFlow sourceRef="TaskQuotationHandling" targetRef="TaskApproveOrder" name="" id="_6-470"/>
    <semantic:sequenceFlow sourceRef="TaskApproveOrder" targetRef="GatewayOrderApprovedDecision" name="" id="_6-500"/>
    <semantic:sequenceFlow sourceRef="GatewayOrderApprovedDecision" targetRef="ParaSplitOrderAndShipment" name="Approved" id="_6-502"/>
    <semantic:sequenceFlow sourceRef="ParaSplitOrderAndShipment" targetRef="_6-190" name="" id="_6-504"/>
    <semantic:sequenceFlow sourceRef="ParaSplitOrderAndShipment" targetRef="_6-241" name="" id="_6-506"/>
    <semantic:sequenceFlow sourceRef="_6-190" targetRef="ParaJoinOderAndShipment" name="" id="_6-508"/>
    <semantic:sequenceFlow sourceRef="_6-241" targetRef="ParaJoinOderAndShipment" name="" id="_6-532"/>
    <semantic:sequenceFlow sourceRef="ParaJoinOderAndShipment" targetRef="TaskReviewOrder" name="" id="_6-534"/>
    <semantic:sequenceFlow sourceRef="TaskReviewOrder" targetRef="EndProcess" name="" id="_6-536"/>
    <semantic:sequenceFlow sourceRef="GatewayOrderApprovedDecision" targetRef="TerminateProcess" name="" id="_6-552"/>
  </semantic:process>
  <semantic:collaboration id="C1275486198151">
    <semantic:participant name="Buyer" processRef="_6-1" id="Buyer"/>
  </semantic:collaboration>
  <bpmndi:BPMNDiagram documentation="" id="Trisotech.Visio-_6" name="Order Process" resolution="96.00000267028808">
    <bpmndi:BPMNPlane bpmnElement="C1275486198151">
  </bpmndi:BPMNDiagram>
</semantic:definitions>
```

7.2 Sub Process and Call Activity

In this section, we explore the use of Sub Processes (expanded and collapsed) along with Call Activities and show how their content can be depicted in separate diagrams.

7.2.1 Expanded Sub Process Example

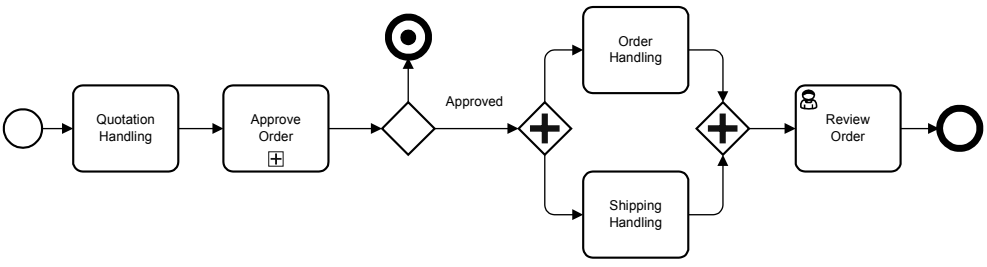
In this example our “Order Process” is depicted with an expanded “Approve Order” Sub Process. The activities within the “Approve Order” Sub Process are part of the parent process. This is a single process depicted in a single diagram.



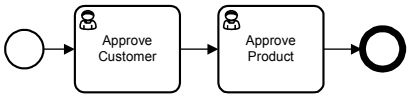

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<semantic:definitions id="_1275486143198" targetNamespace="http://www.trisotech.com/definitions/_1275486143198"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
xmlns:semantic="http://www.omg.org/spec/BPMN/20100524/MODEL">
  <semantic:process isExecutable="false" id="_6">
    <semantic:startEvent name="" id="StartProcess">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Quotation Handling" id="TaskQuotationHandling">
    <semantic:exclusiveGateway gatewayDirection="Diverging" name="" id="GatewayOrderApprovedDecision">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Order  Handling" id="_6-190">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Shipping Handling" id="_6-241">
    <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Review Order" id="TaskReviewOrder">
    <semantic:endEvent name="" id="EndProcess">
    <semantic:subProcess triggeredByEvent="false" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Approver Order" id="SubProcessApproveOrder">
      <semantic:incoming>_6-470</semantic:incoming>
      <semantic:outgoing>_6-500</semantic:outgoing>
      <semantic:startEvent name="" id="SubProcessStart">
      <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Approve Customer" id="TaskApproveCustomer">
      <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Approve Product" id="TaskApproveProduct">
      <semantic:endEvent name="" id="SubProcessEnd">
      <semantic:sequenceFlow sourceRef="SubProcessStart" targetRef="TaskApproveCustomer" name="" id="_6-472"/>
      <semantic:sequenceFlow sourceRef="TaskApproveCustomer" targetRef="TaskApproveProduct" name="" id="_6-474"/>
      <semantic:sequenceFlow sourceRef="TaskApproveProduct" targetRef="SubProcessEnd" name="" id="_6-476"/>
    </semantic:subProcess>
    <semantic:parallelGateway gatewayDirection="Diverging" name="" id="ParaSplitOrderAndShipment">
    <semantic:parallelGateway gatewayDirection="Converging" name="" id="ParaJoinOderAndShipment">
    <semantic:endEvent name="" id="TerminateProcess">
    <semantic:sequenceFlow sourceRef="StartProcess" targetRef="TaskQuotationHandling" name="" id="_6-468"/>
    <semantic:sequenceFlow sourceRef="TaskQuotationHandling" targetRef="SubProcessApproveOrder" name="" id="_6-470"/>
    <semantic:sequenceFlow sourceRef="SubProcessApproveOrder" targetRef="GatewayOrderApprovedDecision" name="" id="_6-500"/>
    <semantic:sequenceFlow sourceRef="GatewayOrderApprovedDecision" targetRef="ParaSplitOrderAndShipment" name="Approved" id="_6-502"/>
    <semantic:sequenceFlow sourceRef="ParaSplitOrderAndShipment" targetRef="_6-190" name="" id="_6-504"/>
    <semantic:sequenceFlow sourceRef="ParaSplitOrderAndShipment" targetRef="_6-241" name="" id="_6-506"/>
    <semantic:sequenceFlow sourceRef="_6-190" targetRef="ParaJoinOderAndShipment" name="" id="_6-508"/>
    <semantic:sequenceFlow sourceRef="_6-241" targetRef="ParaJoinOderAndShipment" name="" id="_6-532"/>
    <semantic:sequenceFlow sourceRef="ParaJoinOderAndShipment" targetRef="TaskReviewOrder" name="" id="_6-534"/>
    <semantic:sequenceFlow sourceRef="TaskReviewOrder" targetRef="EndProcess" name="" id="_6-536"/>
    <semantic:sequenceFlow sourceRef="GatewayOrderApprovedDecision" targetRef="TerminateProcess" name="" id="_6-552"/>
  </semantic:process>
  <bpmndi:BPMNDiagram documentation="" id="Trisotech.Visio-_6" name="Order Process" resolution="96.00000267028808">
    <bpmndi:BPMNPlane bpmnElement="_6">
  </bpmndi:BPMNDiagram>
</semantic:definitions>
```

7.2.2 Collapsed Sub Process Example

In this example our “Order Process” is depicted with a collapsed “Approve Order” Sub Process.



While the content (or details) of the “Approve Order” Sub Process is depicted on a separate diagram.



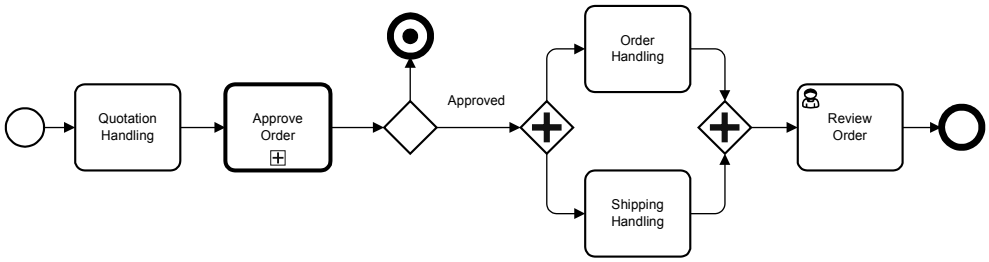
This is a single process depicted into two diagrams: one diagram for the parent process and one diagram for the sub process.

Note that both expanded and collapsed depictions are visual variations of the same single “Order Process”.

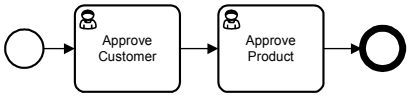
```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<semantic:definitions id="_1275486113839" targetNamespace="http://www.trisotech.com/definitions/_1275486113839"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
xmlns:semantic="http://www.omg.org/spec/BPMN/20100524/MODEL">
  <semantic:process isExecutable="false" id="_6">
    <semantic:startEvent name="" id="StartProcess">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Quotation Handling" id="TaskQuotationHandling">
    <semantic:exclusiveGateway gatewayDirection="Diverging" name="" id="GatewayOrderApprovedDecision">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Order  Handling" id="_6-190">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Shipping Handling" id="_6-241">
    <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Review Order" id="TaskReviewOrder">
    <semantic:endEvent name="" id="EndProcess">
    <semantic:parallelGateway gatewayDirection="Diverging" name="" id="ParaSplitOrderAndShipment">
    <semantic:parallelGateway gatewayDirection="Converging" name="" id="ParaJoinOderAndShipment">
    <semantic:endEvent name="" id="TerminateProcess">
    <semantic:subProcess triggeredByEvent="false" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Approve Order" id="SubProcessApproveOrder">
      <semantic:incoming>_6-470</semantic:incoming>
      <semantic:outgoing>_6-500</semantic:outgoing>
      <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Approve Customer" id="TaskApproveCustomer">
      <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Approve Product" id="TaskApproveProduct">
      <semantic:startEvent name="" id="SubProcessStart">
      <semantic:endEvent name="" id="SubProcessEnd">
      <semantic:sequenceFlow sourceRef="SubProcessStart" targetRef="TaskApproveCustomer" name="" id="_0-131"/>
      <semantic:sequenceFlow sourceRef="TaskApproveCustomer" targetRef="TaskApproveProduct" name="" id="_0-133"/>
      <semantic:sequenceFlow sourceRef="TaskApproveProduct" targetRef="SubProcessEnd" name="" id="_0-135"/>
    </semantic:subProcess>
    <semantic:sequenceFlow sourceRef="StartProcess" targetRef="TaskQuotationHandling" name="" id="_6-468"/>
    <semantic:sequenceFlow sourceRef="TaskQuotationHandling" targetRef="SubProcessApproveOrder" name="" id="_6-470"/>
    <semantic:sequenceFlow sourceRef="SubProcessApproveOrder" targetRef="GatewayOrderApprovedDecision" name="" id="_6-500"/>
    <semantic:sequenceFlow sourceRef="GatewayOrderApprovedDecision" targetRef="ParaSplitOrderAndShipment" name="Approved" id="_6-502"/>
    <semantic:sequenceFlow sourceRef="ParaSplitOrderAndShipment" targetRef="_6-190" name="" id="_6-504"/>
    <semantic:sequenceFlow sourceRef="ParaSplitOrderAndShipment" targetRef="_6-241" name="" id="_6-506"/>
    <semantic:sequenceFlow sourceRef="_6-190" targetRef="ParaJoinOderAndShipment" name="" id="_6-508"/>
    <semantic:sequenceFlow sourceRef="_6-241" targetRef="ParaJoinOderAndShipment" name="" id="_6-532"/>
    <semantic:sequenceFlow sourceRef="ParaJoinOderAndShipment" targetRef="TaskReviewOrder" name="" id="_6-534"/>
    <semantic:sequenceFlow sourceRef="TaskReviewOrder" targetRef="EndProcess" name="" id="_6-536"/>
    <semantic:sequenceFlow sourceRef="GatewayOrderApprovedDecision" targetRef="TerminateProcess" name="" id="_6-552"/>
  </semantic:process>
  <bpmndi:BPMNDiagram documentation="" id="Trisotech.Visio-_6" name="Order Process" resolution="96.00000267028808">
    <bpmndi:BPMNPlane bpmnElement="_6">
  </bpmndi:BPMNDiagram>
  <bpmndi:BPMNDiagram documentation="" id="Trisotech.Visio-_0" name="Approve Order" resolution="96.00000267028808">
    <bpmndi:BPMNPlane bpmnElement="SubProcessApproveOrder">
  </bpmndi:BPMNDiagram>
</semantic:definitions>
```

7.2.3 Call Activity Example

In this example our “Order Process” is depicted with a collapsed Call Activity “Approve Order”. This diagram is quite different than the previous example, as here we are introducing the notion of Process re-use. In this case, the “Approve Order” is not a Sub Process of “Order Process” but separate independent process that is called (re-used) within the “Order Process”.



The “Approve Order” Process



We thus have two processes each in their own diagrams (2 processes, 2 diagrams)

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<semantic:definitions id="_1275486058980" targetNamespace="http://www.trisotech.com/definitions/_1275486058980"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
 xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
 xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
 xmlns:semantic="http://www.omg.org/spec/BPMN/20100524/MODEL">
  <semantic:process isExecutable="false" id="_6">
    <semantic:startEvent name="" id="StartProcess">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Quotation Handling" id="TaskQuotationHandling">
    <semantic:exclusiveGateway gatewayDirection="Diverging" name="" id="GatewayOrderApprovedDecision">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Order  Handling" id="_6-190">
    <semantic:task completionQuantity="1" isForCompensation="false" startQuantity="1" name="Shipping Handling" id="_6-241">
    <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Review Order" id="TaskReviewOrder">
    <semantic:endEvent name="" id="EndProcess">
    <semantic:parallelGateway gatewayDirection="Diverging" name="" id="ParaSplitOrderAndShipment">
    <semantic:parallelGateway gatewayDirection="Converging" name="" id="ParaJoinOderAndShipment">
    <semantic:endEvent name="" id="TerminateProcess">
    <semantic:callActivity calledElement="_0" name="Approve Order" id="SubProcessApproveOrder">
    <semantic:sequenceFlow sourceRef="StartProcess" targetRef="TaskQuotationHandling" name="" id="_6-468"/>
    <semantic:sequenceFlow sourceRef="TaskQuotationHandling" targetRef="SubProcessApproveOrder" name="" id="_6-470"/>
    <semantic:sequenceFlow sourceRef="SubProcessApproveOrder" targetRef="GatewayOrderApprovedDecision" name="" id="_6-500"/>
    <semantic:sequenceFlow sourceRef="GatewayOrderApprovedDecision" targetRef="ParaSplitOrderAndShipment" name="Approved" id="_6-502"/>
    <semantic:sequenceFlow sourceRef="ParaSplitOrderAndShipment" targetRef="_6-190" name="" id="_6-504"/>
    <semantic:sequenceFlow sourceRef="ParaSplitOrderAndShipment" targetRef="_6-241" name="" id="_6-506"/>
    <semantic:sequenceFlow sourceRef="_6-190" targetRef="ParaJoinOderAndShipment" name="" id="_6-508"/>
    <semantic:sequenceFlow sourceRef="_6-241" targetRef="ParaJoinOderAndShipment" name="" id="_6-532"/>
    <semantic:sequenceFlow sourceRef="ParaJoinOderAndShipment" targetRef="TaskReviewOrder" name="" id="_6-534"/>
    <semantic:sequenceFlow sourceRef="TaskReviewOrder" targetRef="EndProcess" name="" id="_6-536"/>
    <semantic:sequenceFlow sourceRef="GatewayOrderApprovedDecision" targetRef="TerminateProcess" name="" id="_6-552"/>
  </semantic:process>
  <semantic:process isExecutable="false" id="_0">
    <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Approve Customer" id="TaskApproveCustomer">
    <semantic:userTask implementation="Other" completionQuantity="1" isForCompensation="false" startQuantity="1" name="Approve Product" id="TaskApproveProduct">
    <semantic:startEvent name="" id="SubProcessStart">
    <semantic:endEvent name="" id="SubProcessEnd">
    <semantic:sequenceFlow sourceRef="SubProcessStart" targetRef="TaskApproveCustomer" name="" id="_0-131"/>
    <semantic:sequenceFlow sourceRef="TaskApproveCustomer" targetRef="TaskApproveProduct" name="" id="_0-133"/>
    <semantic:sequenceFlow sourceRef="TaskApproveProduct" targetRef="SubProcessEnd" name="" id="_0-135"/>
  </semantic:process>
  <bpmndi:BPMNDiagram documentation="" id="Trisotech.Visio-_6" name="Order Process" resolution="96.00000267028808">
    <bpmndi:BPMNPlane bpmnElement="_6">
  </bpmndi:BPMNDiagram>
  <bpmndi:BPMNDiagram documentation="" id="Trisotech.Visio-_0" name="Approve Order" resolution="96.00000267028808">
    <bpmndi:BPMNPlane bpmnElement="_0">
  </bpmndi:BPMNDiagram>
</semantic:definitions>
```

8 Nobel Prize Example

8.1 The Nobel Prize Process Scenario

The selection of a Nobel Prize Laureate is a lengthy and carefully executed process. The processes slightly differ for each of the six prizes; the results are the same for each of the six categories.
Following is the description for the Nobel Prize in Medicine. The main actors in the processes for Nomination, Selection and Accepting and Receiving the award are the:

- Nobel Committee for Medicine,
- Nominators,
- Specially appointed experts,
- Nobel Assembly and
- Nobel Laureates.

Each year in September, in the year preceding the year the Prize is awarded, around 3000 invitations or confidential nomination forms are sent out by the Nobel Committee for Medicine to selected Nominators.

The Nominators are given the opportunity to nominate one or more Nominees. The completed forms must be made available to the Nobel Committee for Medicine for the selection of the preliminary candidates.

The Nobel Committee for Medicine performs a first screening and selects the preliminary candidates.

Following this selection, the Nobel Committee for Medicine may request the assistance of experts. If so, it sends the list with the preliminary candidates to these specially appointed experts with the request to assess the preliminary candidates' work.

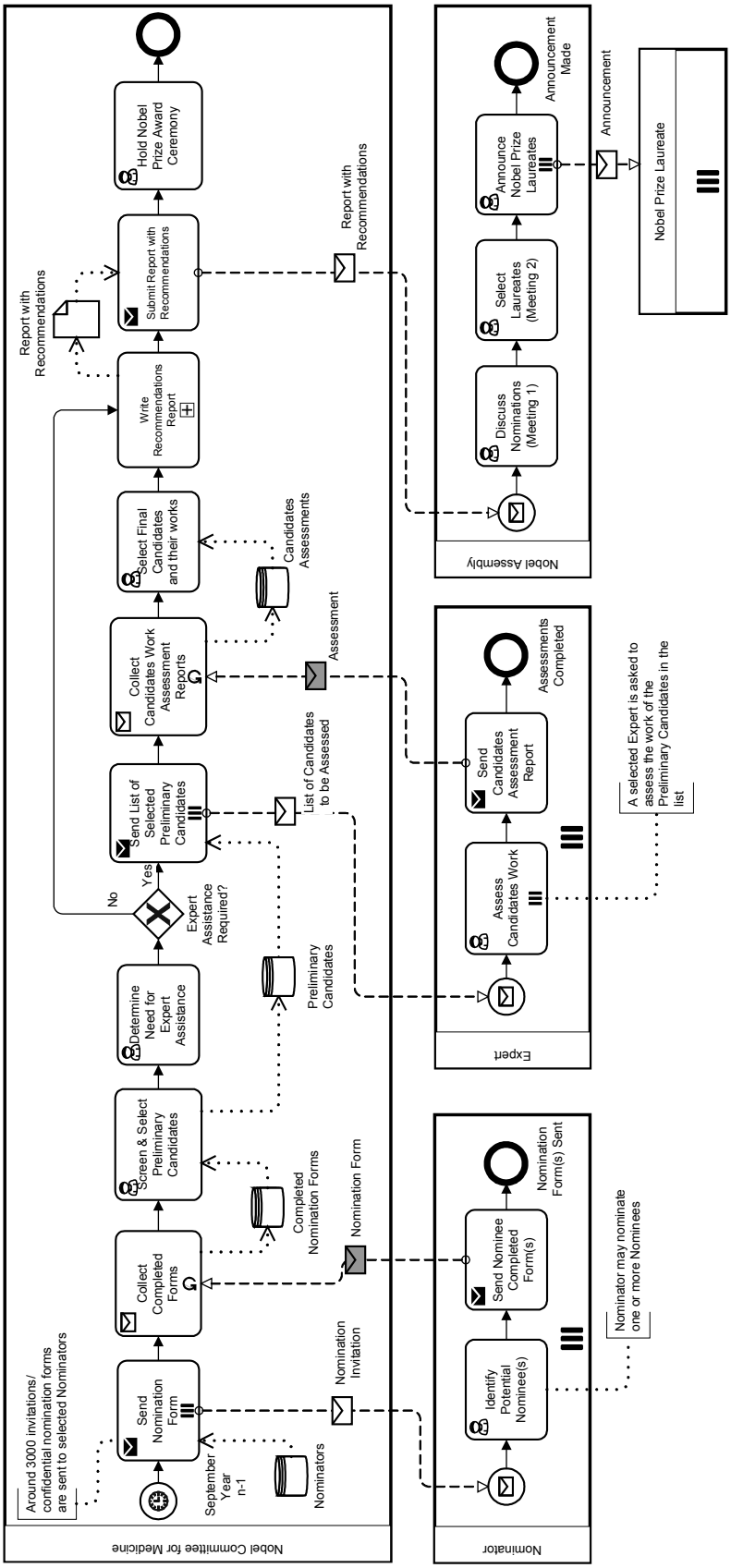
From this, the recommended final candidate laureates and associated recommended final works are selected and the Nobel Committee for Medicine writes the reports with recommendations.

The Nobel Committee for Medicine submits the report with recommendations to the Nobel Assembly. This report contains the list of final candidates and associated works.

The Nobel Assembly chooses the Nobel Laureates in Medicine and associated through a majority vote and the names of the Nobel Laureates and associated works are announced. The Nobel Assembly meets twice for this selection. In the first meeting of the Nobel Assembly the report is discussed. In the second meeting the Nobel Laureates in Medicine and associated works are chosen.

The Nobel Prize Award Ceremony is held in Stockholm.

8.2 The Nobel Prize Process Diagram



9 Travel Booking Example

The purpose of this chapter is to provide an example of in-line event handling via event sub-process constructs.

The process scenario is inspired from figure 10.100 of the BPMN 2.0 Specification document.

9.1 The Travel Booking Scenario

The Travel Agency receives a travel reservation request, including airline transportation and hotel reservation, from a Client.

Following research and evaluation of both flights' and hotel rooms' availability, selected alternatives are packaged and offered to the Client.

The Client has 24 hours to either select a proposed alternative or cancel the request. In case of a cancellation, or after this delay, the Agency updates the Client record to reflect the request cancellation and the Client is notified.

When a selection is made, the Client is asked to provide the Credit Card information. Again, the Client has 24 hours to provide this information or the request is canceled via the same activities stated before (update and notification).

Having received the Credit Card information, the booking activities take place:

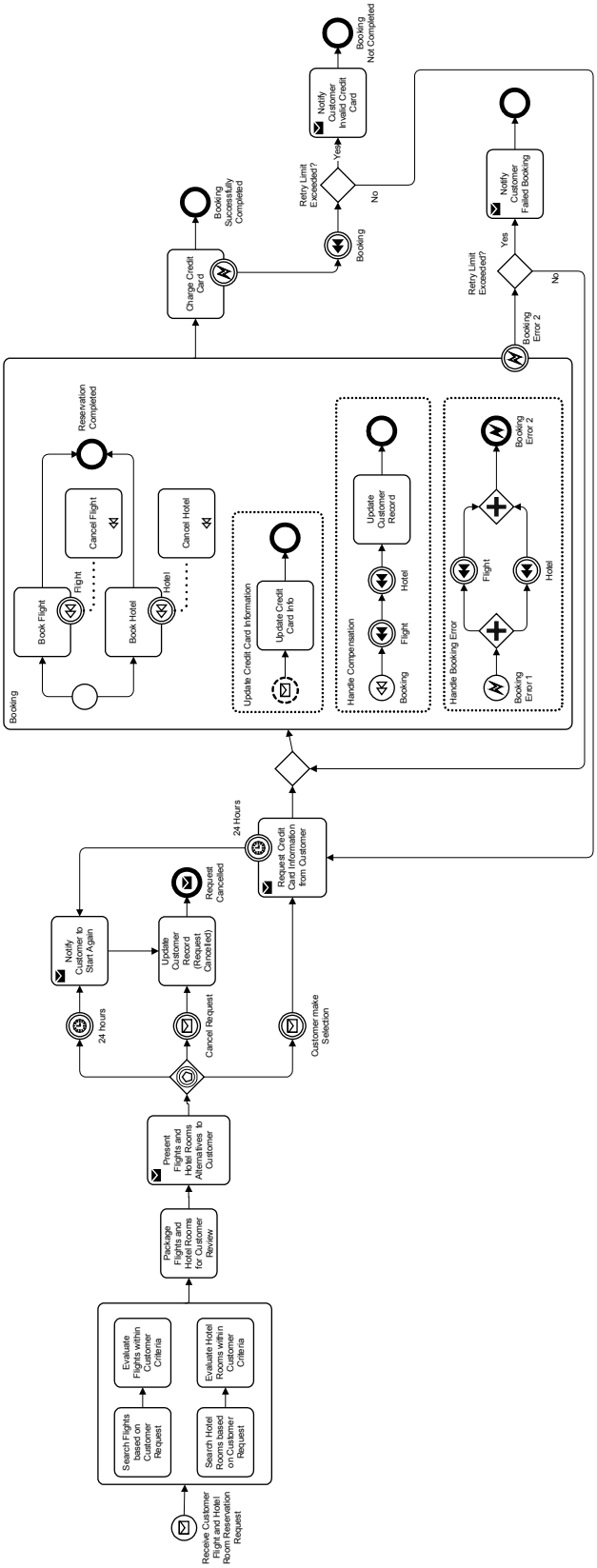
The flight and the hotel room are booked. Measures are taken to insure reservations reversals if problems occur in the booking and payment activities. The Client is also entitled to provide the Agency with Credit Card Information modifications before the booking is completed. Such information will be saved in its record.

If an error arises during the booking activities, the flight and hotel room reservations are reversed and the Client record is updated. The booking is tried again as long as the booking retry limit is not exceeded.

Following successful booking the Reservations are charged on the Client's Credit Card and the process stops following successful confirmation. If an error occurs during this activity the flight and hotel room reservation are reversed. The Client is asked again for the Credit Card Information and the booking is tried again as long as the payment processing retry limit is not exceeded.

In both cases, following the error, when the retry limit is exceeded, the Client is notified and the process stops.

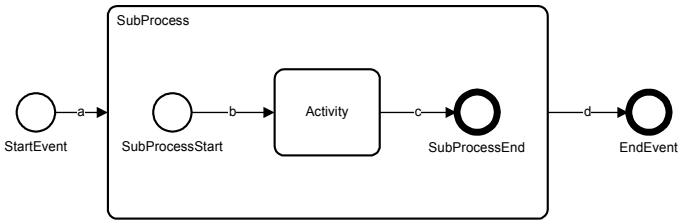
9.2 The Travel Booking Diagram



10 Examples from Diagram Interchange Chapter

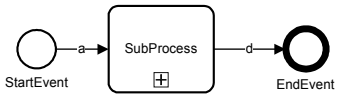
The purpose of this chapter is to provide a subset of the diagrams used into the Notation and Diagrams chapter of the BPMN 2.0 specification along with their serializations. The complete serializations of the herein provided diagrams can be found in the accompanying machine-readable files.

10.1 Expanded Sub Process Example



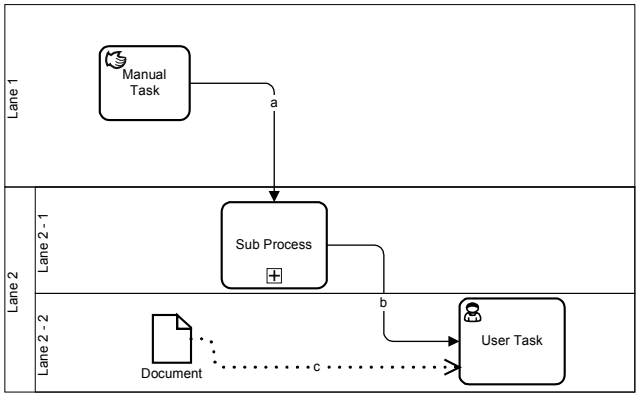
10.2 Collapsed Sub Process Example

10.2.1 Process Diagram

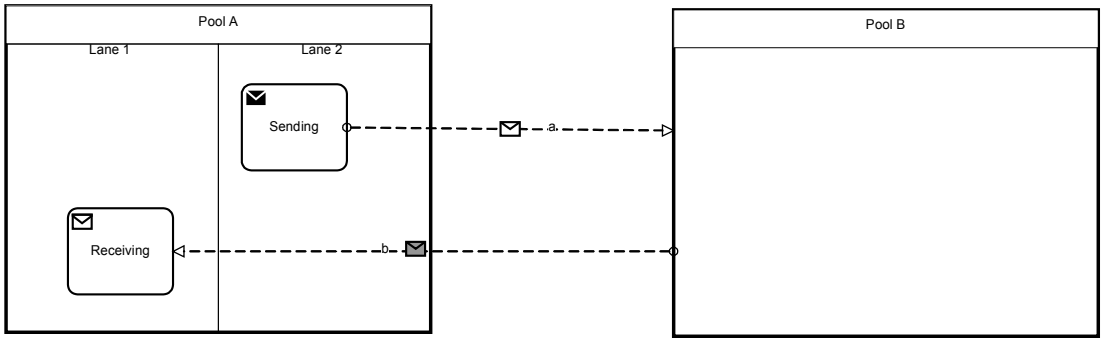


10.2.2 Sub Process Diagram

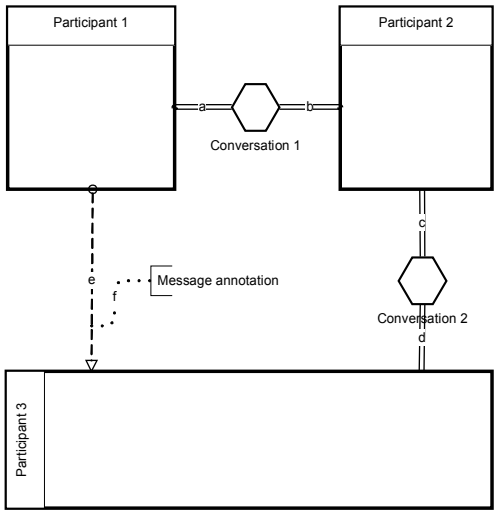
10.3 Multiple Lanes and Nested Lanes Example



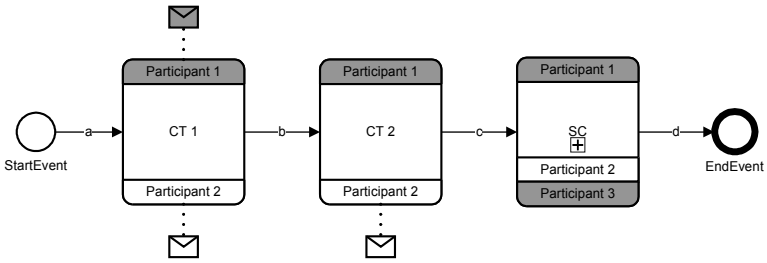
10.4 Vertical Collaboration Example



10.5 Conversation Example



10.6 Choreography Example



11 Correlation Example

This example illustrates the usage of two concepts, namely correlations and definitional collaborations. It introduces a collaboration between three participants - Seller, Buyer and Shipper - and the Seller process that interacts with the two other participants. The SellerCollab collaboration is defined as definitional collaboration for the Seller process, as it specifies all Participants the Process interacts with. It is used to derive which individual service, Send Task or Receive Task, is connected to which Participant through Message Flow and associated correlation information. Moreover, this example illustrates definition of key-based correlations. See Annex A for the according XML Schema and WSDL description.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions id="def"
         targetNamespace="http://www.example.org/Processes/sellerProcess"
         typeLanguage="http://www.w3.org/2001/XMLSchema"
         expressionLanguage="http://www.w3.org/1999/XPath"
         xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:myData="http://www.example.org/Messages"
         xmlns:tns="http://www.example.org/Processes/sellerProcess" >

   <!--    Structures and Messages -->
   <import importType="http://www.w3.org/2001/XMLSchema"
         location="DataDefinitions.xsd"
         namespace="http://www.example.org/Messages"/>
   <import importType="http://schemas.xmlsoap.org/wsdl/"
         location="Interfaces.wsdl"
         namespace="http://www.example.org/Messages"/>

   <itemDefinition id="itemRFQMessage" structureRef="myData:rfqRequest">
      <!-- Single part message -->
   </itemDefinition>
   <itemDefinition id="itemQuoteMessage" structureRef="myData:rfqResponse">
      <!-- Single part message -->
   </itemDefinition>
   <itemDefinition id="itemFaultMessage" structureRef="myData:rfqFault">
      <!-- Single part message -->
   </itemDefinition>
   <itemDefinition id="itemOrderRequest" structureRef="myData:orderRequest">
      <!-- Multi part message -->
   </itemDefinition>
   <itemDefinition id="itemOrderResponse" structureRef="myData:orderResponse">
      <!-- Multi part message -->
   </itemDefinition>

   <itemDefinition id="itemShippingRequest" structureRef="myData:shippingRequest">
      <!-- Multi part message -->
   </itemDefinition>

   <itemDefinition id="itemShippingResponse" structureRef="myData:shippingResponse">
      <!-- Multi part message -->
   </itemDefinition>

   <message id="msgRFQ" name="RFQ Message" itemRef="tns:itemRFQMessage"/>
   <message id="msgQuote" name="Quote Message" itemRef="tns:itemQuoteMessage"/>
   <message id="msgFault" name="Fault Message" itemRef="tns:itemFaultMessage"/>
   <message id="msgOrderData" name="Order Data Message" itemRef="tns:itemOrderRequest"/>
   <message id="msgOrderConfirmation" name="Order Confirmation Message" itemRef="tns:itemOrderResponse"/>
   <message id="msgShippingData" name="Shipping Data Message" itemRef="tns:itemShippingRequest"/>
   <message id="msgShippingConfirmation" name="Shipping Confirmation Message" itemRef="tns:itemShippingResponse"/>

   <partnerEntity id="theSeller" name="The Seller">
         <participantRef>tns:seller</participantRef>
   </partnerEntity>
   <partnerRole id="aBuyer" name="A Buyer">
          <participantRef>tns:buyer</participantRef>
   </partnerRole>
   <partnerRole id="aShipper" name="A Shipper">
          <participantRef>tns:shipper</participantRef>
   </partnerRole>

   <correlationProperty id="propQuoteID" name="Property Quote ID" type="xsd:string">
      <correlationPropertyRetrievalExpression messageRef="tns:msgRFQ">
        <messagePath>/request/quoteID</messagePath>
```

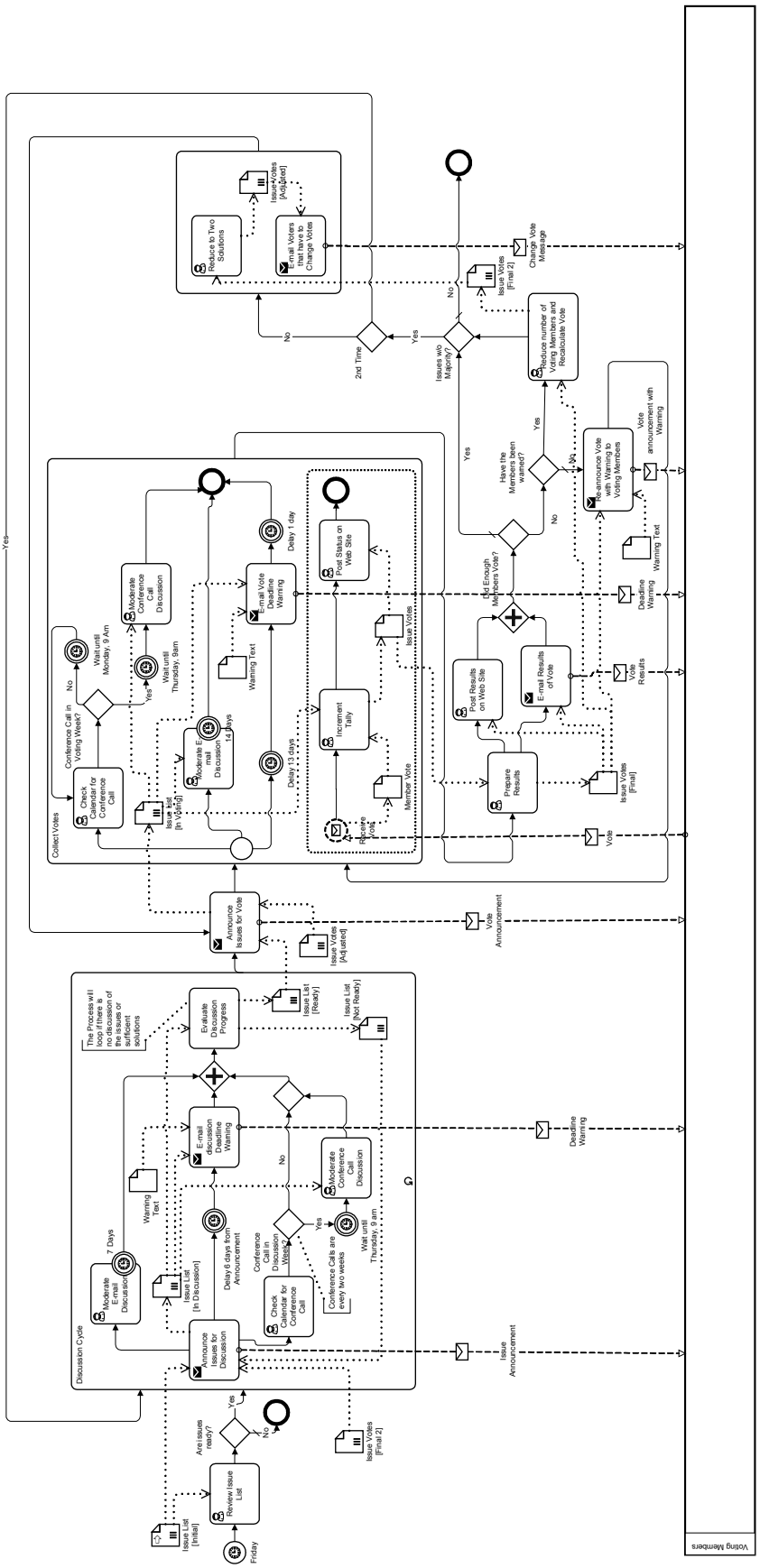
```
      </correlationPropertyRetrievalExpression>
      <correlationPropertyRetrievalExpression messageRef="tns:msgQuote">
         <messagePath>/response/quoteID</messagePath>
      </correlationPropertyRetrievalExpression>
      <correlationPropertyRetrievalExpression messageRef="tns:msgFault">
         <messagePath>/fault/quoteID</messagePath>
      </correlationPropertyRetrievalExpression>
      <correlationPropertyRetrievalExpression messageRef="tns:msgOrderData">
         <messagePath>/priceQuotationRef</messagePath>
      </correlationPropertyRetrievalExpression>
   </correlationProperty>
   <correlationProperty id="propCustomerID" name="Property Customer ID" type="xsd:string">
      <correlationPropertyRetrievalExpression messageRef="tns:msgOrderData">
         <messagePath>/customer/id</messagePath>
      </correlationPropertyRetrievalExpression>
      <correlationPropertyRetrievalExpression messageRef="tns:msgOrderConfirmation">
         <messagePath>/customerID</messagePath>
      </correlationPropertyRetrievalExpression>
   </correlationProperty>
   <correlationProperty id="propOrderID" name="Property Order ID" type="xsd:string">
      <correlationPropertyRetrievalExpression messageRef="tns:msgOrderData">
         <messagePath>/order/orderID</messagePath>
      </correlationPropertyRetrievalExpression>
      <correlationPropertyRetrievalExpression messageRef="tns:msgOrderConfirmation">
         <messagePath>/order/orderID</messagePath>
      </correlationPropertyRetrievalExpression>
      <correlationPropertyRetrievalExpression messageRef="tns:msgShippingData">
         <messagePath>/order/orderID</messagePath>
      </correlationPropertyRetrievalExpression>
      <correlationPropertyRetrievalExpression messageRef="tns:msgShippingConfirmation">
         <messagePath>/order/orderID</messagePath>
      </correlationPropertyRetrievalExpression>
   </correlationProperty>

   <collaboration id="sellerCollab">
      <participant id="seller" name="Seller" processRef="tns:sellerProcess">
         <interfaceRef>tns:sellerServiceInterface</interfaceRef>
      </participant>
      <participant id="buyer" name="Buyer"/>
      <participant id="shipper" name="Shipper">
         <interfaceRef>tns:shipperServiceInterface</interfaceRef>
      </participant>
      <messageFlow id="mf1" messageRef="tns:msgRFQ" sourceRef="tns:buyer" targetRef="tns:receiveQuoteRequest"/>
      <messageFlow id="mf2" messageRef="tns:msgQuote" sourceRef="tns:sendQuote" targetRef="tns:buyer"/>
      <messageFlow id="mf3" messageRef="tns:msgFault" sourceRef="tns:sendFault" targetRef="tns:buyer"/>
      <messageFlow id="mf4" messageRef="tns:msgOrderData" sourceRef="tns:buyer"
        targetRef="tns:receiveOrderRequest"/>
      <messageFlow id="mf5" messageRef="tns:msgOrderConfirmation" sourceRef="tns:sendOrderResponse"
        targetRef="tns:buyer"/>
      <messageFlow id="mf6" messageRef="tns:msgShippingData" sourceRef="tns:sendShippingRequest"
        targetRef="tns:shipper"/>
      <messageFlow id="mf7" messageRef="tns:msgShippingConfirmation" sourceRef="tns:shipper"
        targetRef="tns:receiveShippingConfirmation"/>

      <!--   Conversations -->
      <conversation id="conversationQuoteRequest">
         <messageFlowRef>tns:mf1</messageFlowRef>
         <messageFlowRef>tns:mf2</messageFlowRef>
         <messageFlowRef>tns:mf3</messageFlowRef>
         <messageFlowRef>tns:mf4</messageFlowRef>
         <correlationKey id="correlQuote" name="Quote Correlation Key">
            <correlationPropertyRef>tns:propQuoteID</correlationPropertyRef>
         </correlationKey>
      </conversation>
      <conversation id="conversationOrderHandling">
         <messageFlowRef>tns:mf4</messageFlowRef>
         <messageFlowRef>tns:mf5</messageFlowRef>
         <correlationKey id="correlOrder" name="Order Correlation Key">
            <correlationPropertyRef>tns:propCustomerID</correlationPropertyRef>
            <correlationPropertyRef>tns:propOrderID</correlationPropertyRef>
         </correlationKey>
      </conversation>
      <conversation id="conversationShipmentRequest">
         <messageFlowRef>tns:mf6</messageFlowRef>
         <messageFlowRef>tns:mf7</messageFlowRef>
         <correlationKey id="correlShipment" name="Shipment Correlation Key">
            <correlationPropertyRef>tns:propOrderID</correlationPropertyRef>
         </correlationKey>
      </conversation>
   </collaboration>

   <!--    Interfaces -->
```

```
<!-- The interface of the Seller Process -->
<interface id="sellerServiceInterface" name="Seller Service Interface">
   <operation id="requestQuoteOp" name="Request Quote Operation">
      <inMessageRef>tns:msgRFQ</inMessageRef>
      <outMessageRef>tns:msgQuote</outMessageRef>
      <errorRef>tns:msgFault</errorRef>
   </operation>
   <operation id="orderOp" name="Order Operation">
      <inMessageRef>tns:msgOrderData</inMessageRef>
      <outMessageRef>tns:msgOrderConfirmation</outMessageRef>
   </operation>
</interface>
<interface id="shipperServiceInterface" name="Shipper Service Interface">
   <operation id="requestShippingOp" name="Request Shipping Operation">
      <inMessageRef>tns:msgShippingData</inMessageRef>
      <outMessageRef>tns:msgShippingConfirmation</outMessageRef>
   </operation>
</interface>

<!--    Process Definition -->

<process id="sellerProcess" name="Seller process"
      definitionalCollaborationRef="tns:sellerCollab">

   <!--Receive quote request message from caller.-->
   <receiveTask id="receiveQuoteRequest" name="Receive Quote Request"
              instantiate="true"
              messageRef="tns:msgRFQ"
              operationRef="tns:requestQuoteOp"/>

   <sequenceFlow targetRef="decision1" sourceRef="receiveQuoteRequest"/>
   <!--Decide whether quote is available and can be returned, or not.
              The actual processing logic is omitted from the example. -->

   <exclusiveGateway id="decision1" gatewayDirection="Mixed"
              default="noQuote"/>
   <sequenceFlow id="quote" targetRef="sendQuote" sourceRef="decision1">
      <conditionExpression>Quote available and okay.</conditionExpression>
   </sequenceFlow>
   <sequenceFlow id="noQuote" targetRef="sendFault" sourceRef="decision1"/>

   <!-- Respond successful quote back to caller. -->
   <sendTask id="sendQuote" name="Send Quote"
     messageRef="tns:msgQuote"
              operationRef="tns:requestQuoteOp"/>

   <sequenceFlow targetRef="eventWait" sourceRef="sendQuote"/>

   <!-- This is a reply, so use same service reference and operation as in associated receive. -->
   <sendTask id="sendFault" name="Send Fault"
     messageRef="tns:msgFault"
              operationRef="tns:requestQuoteOp"/>

       <receiveTask id="receiveNewQuoteRequest" name="Receive New Quote"
              messageRef="tns:msgRFQ"
              operationRef="tns:requestQuoteOp"/>
   <sequenceFlow targetRef="decision1" sourceRef="receiveNewQuoteRequest"/>

   <!-- Respond error back to caller -->

   <sequenceFlow targetRef="eventWait" sourceRef="sendFault"/>

   <!-- Wait for another quote request, an order, or a timeout -->
   <eventBasedGateway id="eventWait" gatewayDirection="Mixed"/>
   <sequenceFlow targetRef="receiveNewQuoteRequest" sourceRef="eventWait"/>
   <sequenceFlow targetRef="receiveOrderRequest" sourceRef="eventWait"/>
   <sequenceFlow targetRef="timeout" sourceRef="eventWait"/>
   <!-- Timeout and end -->
   <intermediateCatchEvent id="timeout">
      <timerEventDefinition>
         <timeDate>PD4h</timeDate>
      </timerEventDefinition>
   </intermediateCatchEvent>
   <sequenceFlow targetRef="end1" sourceRef="timeout"/>
   <endEvent id="end1"/>

   <!-- Receive an order message-->
   <receiveTask id="receiveOrderRequest" name="Receive Order Request"
    messageRef="tns:msgOrderData"
    operationRef="tns:orderOp" />

   <sequenceFlow targetRef="fork" sourceRef="receiveOrderRequest"/>
```

```
      <parallelGateway id="fork" gatewayDirection="Diverging"/>
      <sequenceFlow targetRef="sendOrderResponse" sourceRef="fork"/>
      <sequenceFlow targetRef="sendShippingRequest" sourceRef="fork"/>

      <!-- Send order confirmation -->
      <sendTask id="sendOrderResponse" name="Send Order Response"
        messageRef="tns:msgOrderConfirmation"
        operationRef="tns:orderOp" />

      <sequenceFlow targetRef="join" sourceRef="sendOrderResponse"/>

      <!-- Trigger Shipping -->
      <sendTask id="sendShippingRequest" name="Send Shipping Request"
        messageRef="tns:msgShippingData"
        operationRef="tns:requestShippingOp"/>

      <sequenceFlow targetRef="receiveShippingConfirmation"
        sourceRef="sendShippingRequest"/>
      <!-- Receive Shipment Notification -->

      <receiveTask id="receiveShippingConfirmation" name="Receive Shipping Confirmation"
                messageRef="tns:msgShippingConfirmation"
                operationRef="tns:requestShippingOp" />

      <sequenceFlow targetRef="join" sourceRef="receiveShippingConfirmation"/>
      <parallelGateway id="join" gatewayDirection="Converging"/>
      <sequenceFlow targetRef="end2" sourceRef="join"/>
      <endEvent id="end2"/>
   </process>
</definitions>
```

12 E-Mail Voting Example

This chapter will provide an example of a business process modeled with BPMN. This example was presented in the BPMN 1.0 specification, but has been updated for BPMN 2.0. The process that will be described is a process used to help develop this notation. It is a process for resolving issues through e-mail votes (see Figure). This Process is small, but fairly complex and will provide examples for many of the features of BPMN, and it will help illustrate that BPMN can handle simple and unusual business processes and still be easily understandable for readers of the Diagram. The sections below will isolate segments of the Process and highlight the modeling features as the workings of the Process is described.


Voting Members
Friday
Review Issue List
Are issues ready?
Yes
No
Issue List [Initial]
Issue List [Final 2]
Discussion Cycle
Moderate Email Discussion
7 Days
Issue List [In Discussion]
Warning Text
Announce Issues for Discussion
E-mail discussion Deadline Warning
Delay 6 days from Announcement
Conference Call in Discussion Week?
Check Calendar for Conference Call
Conference Calls are every two weeks
Moderate Conference Call Discussion
Wait until Thursday, 9 am
No
Yes
Evaluate Discussion Progress
The Process will loop if there is no discussion of the issues or sufficient solutions
Issue List [Ready]
Issue List [Not Ready]
Deadline Warning
Issue Announcement
Announce Issues for Vote
Issue Votes [Adjusted]
Vote Announcement
Collect Votes
Conference Call in Voting Week?
No
Yes
Wait until Monday, 9 Am
Wait until Thursday, 9am
Moderate Conference Call Discussion
Check Calendar for Conference Call
Issue List [In Voting]
Moderate E-mail Discussion 1 of 2 days
Delay 13 days
E-mail Vote Deadline Warning
Delay 1 day
Warning Text
Receive Votes
Increment Tally
Member Vote
Issue Votes
Post Status on Web Site
Vote
Prepare Results
Post Results on Web Site
E-mail Results of Vote
Issue Votes [Final]
Vote Results
Deadline Warning
Did Enough Members Vote?
Have the Members been asked?
Announce Vote with Warning to Voting Members
Vote announcement with Warning
Warning Text
Reduce number of Voting Members and Recalculate Vote
Issues w/o Majority?
Yes
No
Issue Votes [Final 2]
Change Vote Message
2nd Time?
No
Yes
Reduce to Two Solutions
E-mail Voters that have to Change Votes
Issue Votes [Adjusted]

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

The Issue List Manager will review the list and determine if there are any issues that are ready for going through the discussion and voting cycle. Then a Decision must be made. If there are no issues ready, then the Process is over for that week--to be taken up again the following week. If there are issues ready, then the Process will continue with the discussion cycle. The "Discussion Cycle" Sub-Process is the first activity after the "Any issues ready?" Decision and this Sub-Process has two incoming Sequence Flow, one of which originates from a downstream Decision and is thus part of a loop. It is one of a set of four (4) complex loops that exist in the Process. The contents of the "Discussion Cycle" Sub-Process and the activities that follow will be described below.

12.1 The First Sub-Process

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

The top parallel path in the figure starts with a long-running Task, "Moderate E-mail Discussion," that has a Timer Intermediate Event attached to its boundary. The "Moderate E-Mail Discussion" Task will never actually be completed normally in this model, but will be interrupted by the Timer Intermediate Event.

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

The bottom parallel path of the fork contains more than one object, first of which is Task where the issue list manager checks the calendar to see if there is a conference call this week. The output of the Task will be an update to the variable "ConCall" (not seen), which will be true or false. After the Task, an Exclusive Gateway with its two Gates follows. The "default" Flow connects directly to an merging Exclusive Gateway. A merging Exclusive Gateway is used in this situation because the next object is a joining Parallel Gateway (the diamond with the cross in the center) that is used to synchronize the three (3) parallel paths. If the merging Gateway was not used and both Sequence Flow connected to the Parallel Gateway, the Process would have been stuck at the Parallel Gateway that would wait for a Token to arrive from each of the incoming Sequence Flow. The "Yes" Sequence Flow will have a *condition* that checks the value of the "ConCall" variable (set in the previous Task) to see if there will be a conference call during the week. If so, the Timer Intermediate Event indicates delay, since all conference calls for the working group start at 9am PDT on Thursdays. The Task for moderating the conference call follows the delay, which is followed by the merging Gateway.

The merging Gateway in bottom path, the "Moderate E-mail Discussion" Task, and the "E-Mail Discussion Deadline Warning" Task all flow into a synchronizing Parallel Gateway. This Gateway waits for all three paths to complete before the Process will continue to the next Task, "Evaluate Discussion Progress." The Issue List Manager will review the status of the issues and the discussions during the past week and decide if the discussions are over. The "DiscussionOver" variable (not seen) will be set to TRUE or FALSE, depending on this evaluation. If the variable is set to FALSE, then the whole Sub-Process will be repeated, since it has looping set and the loop condition that will test the "DiscussionOver" variable.

12.2 The Second Sub-Process

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

The “Collect Votes” Sub-Process follows the Task, and is also a target of one of the looping Sequence Flow. This Sub-Process is basically a set of three (3) parallel paths that extend from the beginning to the end of the Sub-Process. In addition, there is a non-interrupting Event Sub-Process that is used to receive the votes from the voting members as they come in.

The first branch of the fork leads to a Decision that determines whether or not a conference call will occur during the upcoming week, after the Working Group’s schedule has been checked. Basically, if there was a call last week, then there will not be a call this week and vice versa. If there is no call, then there is a Timer Intermediate Event that is set to wait until the next Monday, then the path loops back. The appropriate variable that was updated in the “Discussion Cycle” Process will be used again.

The second and third branches of the forks work the same way as the similar activities in the “Discussion Cycle” Sub-Process, except that it will last two weeks. However, since the branches lead to an End Event instead of a Parallel Gateway, a merging Exclusive Gateway is not needed (the necessary synchronization will be done by the End Event).

The Event Sub-Process will accept votes from the voting members throughout the two weeks that the “Collect Votes” Sub-Process runs. The policy of the working group is that voting members can vote more than once on an issue; that is, they can change their mind as many times as they want throughout the entire two weeks. The Message Start Event triggers the performance of the Event Sub-Process. It is of the non-interrupting type so that multiple votes can be collected during the two weeks. As part of this, an incoming Message Flow is seen from the “Voting Members” Pool to the “Receive Vote” Start Event. Within the Event Sub-Process are Two Tasks that follow the start. First, a Task will prepare all the voting results, then a Task will send the results to the voting members.

12.3 The End of the Process

The last section of the Process includes a complex set of Decisions and loops. First a set of Tasks will prepare the voting results, email them to the voting members, and post them on a web site. The first Decision, “Did Enough Members Vote?,” is necessary since two-thirds of the voting members are required to approve any solution to an issue. If less than two-thirds of the voting members cast votes, which sometimes happens, the issues can’t be resolved. This Decision is followed by another Decision for both of its Alternatives. The “No” Alternative is followed by the “Have the Members been Warned?” Decision. If a voting member misses a vote, they are warned. If they miss a second vote, they lose their status as a voting member and the voting percentages are recalculate through a Task (“Reduce number of Voting Members and Recalculate Vote”). If they haven’t yet been warned, then a warning is sent and the voting cycle is repeated. If all issues are resolved, then the Process is done. If not, then another Decision is required. The voting is given two chances before it goes back to another cycle of discussion. The first time will see a reduction of the number of solutions to the two most popular based on the vote (more if there are ties). Some voting members will have to change their votes just because their selected solution is no longer valid. These two activities are placed in a Sub-Process to show how a Sub-Process without Start and End Events can be used to create a simple set of parallel activities. Informally, this is called a “parallel box.” It is not a special object, but another use of Sub-Processes. For simple situations, it can be used to show a set of parallel activities without the extra clutter of a lot of Sequence Flow. In actuality, these two Tasks cannot actually be done in parallel, but they are modeled this way to highlight the optional use of Start and End Events. After the parallel box, the flow loops back to the “Collect Votes” Sub-Process. If there already has been two cycles of voting, then the process Flow back to the “Decision Cycle” Sub-Process.

Annex A: XML Serializations for all presented Models

(informative)

A.1 Machine-readable XML Serializations

The XML serializations for all models are provided in machine-readable form as a separate zip file, which has the OMG Document Number dtc/2010-06-03 and is available for download at http://www.omg.org/spec/BPMN/2.0/examples/ZIP.