

University of Crete  
School of Sciences and Engineering  
Computer Science Department

**DESIGNING, IMPLEMENTING AND EXECUTING  
CHOREOGRAPHIES AND ORCHESTRATIONS  
OVER A BPMN 2 ENGINE**

**BY**

***MICHAIL DIMITRIOU***

Master's Thesis

Heraklion, November 2011



University of Crete  
School of Sciences and Engineering  
Computer Science Department

**DESIGNING, IMPLEMENTING AND EXECUTING  
CHOREOGRAPHIES AND ORCHESTRATIONS OVER A BPMN 2  
ENGINE**

BY

***MICHAIL DIMITRIOU***

*A thesis submitted in partial fulfillment  
of the requirements for the degree of  
Master of Science*

Author:



---

Michail Dimitriou, Computer Science Department

*Board of enquiry:*

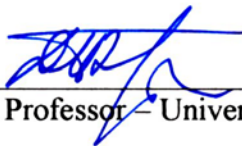
Supervisor:



---

Christos Nikolaou, Professor - University of Crete

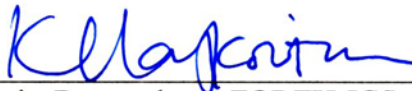
Member:



---

Dimitrios Plexousakis, Professor - University of Crete

Member:



---

Kostantinos Magoutis, Researcher - FORTH-ICS

Approved by:



---

Angelos Bilas, Associate Professor - University of Crete, Chairman of the Graduate  
Studies Committee



DESIGNING, IMPLEMENTING AND EXECUTING CHOREOGRAPHIES AND  
ORCHESTRATIONS OVER A BPMN 2 ENGINE

**Dimitiou Michail**

**Master's Thesis**

**University of Crete**

**Computer Science Department**

## **Abstract**

For decades now business managers have been using workflows to describe and study business processes. Soon after such workflows started appearing in the designing of Information Systems and the two worlds started their parallel evolution. Tools and standards were developed to design and study these models yet most of them were focusing on the one side of the coin. With the arrival of web services the similarity of a web service task to a business task and a business process to a service composition became obvious and the attempt to merge the world of management and IT began.

With web service composition in mind several metamodels were proposed (WSFL, XLANG, BPML). With IBM and Microsoft leading the way **Business Process Execution Language (BPEL)** evolved from the above and became the standard for service composition and Business process execution. BPEL although excellent for machine readability (execution) and automated processes, proved too complicated for non developers and as no graphical representation was in mind when developed the management community sought after a standard for designing and studying models. Later **Business Process Modeling Notation (BPMN) became the preferred designing metamodels for describing and specifying business tasks in a business process model**. BPMN contrary to BPEL although easily human readable and with graphical representation wasn't strict enough to be executable, as a result the two standards coexisted although mapping between them was required in order to transcend from design to execution. With the evolution of the field and the new notions orchestrations as independent processes, choreographies as the collaboration and interaction of multiple orchestrations, human tasks and several more the development of a new notation looked necessary.

The result as of 2010 was BPMN 2. This notation can be graphically represented and is strict enough to be executed can describe choreographies and human tasks and most importantly is easily extendable and configurable. In This Thesis we try to combine and refine some existing tools to create an infrastructure where someone can easily design and execute orchestrations and choreographies using BPMN 2.



**ΣΧΕΔΙΑΣΜΟΣ, ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΕΚΤΕΛΕΣΗ ΧΟΡΟΓΡΑΦΙΩΝ  
ΚΑΙ ΕΝΟΡΧΗΣΤΡΩΣΕΩΝ ΠΑΝΩ ΣΕ ΜΙΑ BPMN 2 ΜΗΧΑΝΗ.**

*Δημητρίου Μιχαήλ*  
*Master's Thesis*  
*Πανεπιστήμιο Κρήτης*  
*Τμήμα Επιστήμης Υπολογιστών*

Δεκαετίες τώρα στελέχη επιχειρήσεων χρησιμοποιούν workflows για την περιγραφή και μελέτη επιχειρηματικών διαδικασιών. Σύντομα αυτά άρχισαν να εμφανίζονται στο σχεδιασμό των Πληροφοριακών Συστημάτων και οι δύο κόσμους άρχισαν την παράλληλη εξέλιξή τους. Διάφορα εργαλεία και πρότυπα αναπτύχθηκαν για το σχεδιασμό και τη μελέτη αυτών των μοντέλων αλλά τα περισσότερα από αυτά επικεντρώνονταν στη μία πλευρά του νομίσματος. Με την έλευση των διαδικτυακών υπηρεσιών η ομοιότητα ενός web service με μιας επιχειρησιακής εργασίας και μια επιχειρηματική διαδικασία με μια σύνθεση από web Services έγινε φανερή και η προσπάθεια να συγχωνευτεί ο κόσμος του Management και του IT ξεκίνησε.

Με τη σύνθεση υπηρεσιών Ιστού σαν βάση προτάθηκαν πολλά μεταμοντέλα όπως (WSFL, XLANG, BPML). Με την IBM και η Microsoft σαν οδηγούς εξελίχθηκε η **Business Process Execution Language (BPEL)** και έγινε το πρότυπο για τη σύνθεση υπηρεσιών και εκτέλεσης Επιχειρησιακών διαδικασιών. Η BPEL αν και εξαιρετική για ανάγνωση (εκτέλεση) από μία μηχανή και για αυτοματοποιημένες διαδικασίες, αποδείχθηκε υπερβολικά περίπλοκη για τους μη προγραμματιστές και δεδομένου ότι δεν είχε κατά νου την γραφική αναπαράσταση όταν αναπτύχθηκε η κοινότητα του Management αναζήτησε ένα διαφορετικό πρότυπο για το σχεδιασμό και τη μελέτη μοντέλων. Αργότερα Η **Business Process Modeling Notation (BPMN)** έγινε το προτιμώμενο μοντέλο για τον σχεδιασμό και την περιγραφή επιχειρησιακών εργασιών και διαδικασιών. Η **BPMN** σε αντίθεση με την **BPEL** αν και εύκολα αναγνώσιμη από τον άνθρωπο και με γραφική παράσταση δεν ήταν αρκετά αυστηρά δομημένη ώστε να είναι εκτελέσιμη, με αποτέλεσμα τα δύο πρότυπα να συνυπάρχουν, αν και χαρτογράφηση μεταξύ τους ήταν απαραίτητη ώστε να μεταβούμε από το σχεδιασμό στην εκτέλεση. Με την εξέλιξη του τομέα και τις νέες έννοιες της ενορχήστρωσης σαν μία ανεξάρτητη διαδικασία, της χορογραφίας σαν την συνεργασία και αλληλεπίδραση πολλών ενορχηστρώσεων, τις ανθρώπινες εργασίες και αρκετές ακόμα, η ανάπτυξη μιας νέας σημειογραφίας φαινόταν απαραίτητη.

Το αποτέλεσμα από το 2010 ήταν Η **BPMN 2**. Αυτή η σημειογραφία μπορεί να αναπαρασταθεί γραφικώς και είναι αρκετά αυστηρή ώστε να εκτελεστεί, μπορεί να περιγράψει χορογραφίες και τις ανθρώπινες εργασίες και κυρίως είναι εύκολα επεκτάσιμη και παραμετροποιήσιμη. Στην παρούσα εργασία προσπαθούμε να συνδυάσουμε και να τελειοποιήσουμε κάποια υπάρχοντα εργαλεία για τη δημιουργία μιας υποδομής, όπου κάποιος θα μπορεί εύκολα να σχεδιάζει και να εκτελεί ενορχηστρώσεις και χορογραφίες χρησιμοποιώντας **BPMN 2**.



## Acknowledgements

First of all I would like to thank my supervisor, Mr. Christos Nikolaou for the cooperation we had over all these years, for his help and guidance and for giving me the opportunity to develop and implement my ideas.

I would also like to thank Mr. Dimitris Plexousakis and Mr. Kostas Magoutis for being in my board of enquiry and for reading my work.

Furthermore I'd like to thank all my friends in Heraklion for the moments we had the years of my stay, my colleagues in the Transformation and Services Laboratory for the cooperation. And of course all my Teachers and peers in the undergraduate and post graduate program of the Computer Science Department of Crete.

I must also thank my Employer Virtual Trip Ltd. and all my colleagues there for the chance to improve and develop my skills and the Job position that supported me financially during my post graduate years.

Finally I my greatest appreciation go to my family that supported me financially and ethically throughout my studies and stood behind my every choice whether right or wrong.



## Table of Contents

<b>ABSTRACT .....</b>	<b>1</b>
<b>ACKNOWLEDGEMENTS .....</b>	<b>3</b>
<b>TABLE OF CONTENTS .....</b>	<b>4</b>
<b>TABLE LIST .....</b>	<b>5</b>
<b>FIGURE LIST .....</b>	<b>6</b>
<b>CODE SNIPPET LIST .....</b>	<b>7</b>
<b>1 INTRODUCTION.....</b>	<b>8</b>
<b>2 BACKGROUND KNOWLEDGE/THEORY .....</b>	<b>8</b>
2.1 WEB SERVICES .....	8
2.2 ORCHESTRATIONS.....	9
2.3 CHOREOGRAPHIES .....	9
2.4 ORCHESTRATIONS AND CHOREOGRAPHIES .....	9
2.5 BMPN .....	9
2.6 BPEL .....	10
2.7 BPEL AND BPMN .....	11
2.8 BPMN 2.....	11
<b>3 THE INFRASTRUCTURE.....</b>	<b>12</b>
3.1 THE BPMN 2 ENGINE .....	12
3.1.1 <i>The engines API and its hierarchy can be seen in the following figure.</i> .....	12
3.2 MySQL DATABASE .....	14
3.3 BPMN 2 DESIGNER TOOL.....	16
3.4 MANAGEMENT WEB APPLICATION .....	18
3.4.1 <i>Architecture and functionalities.</i> .....	22
3.5 SUMMARY.....	25
<b>4 THE FRAMEWORK IN PRACTICE .....</b>	<b>26</b>
4.1 LEARNING THE BPMN 2.0 CONSTRUCTS.....	26
4.1.1 <i>None start event</i> .....	26
4.1.2 <i>None end event</i> .....	27
4.1.3 <i>Sequence flow</i> .....	27
4.1.4 <i>Conditional sequence flow</i> .....	28
4.1.5 <i>Gateways</i> .....	29
4.1.6 <i>User task</i> .....	34
4.1.7 <i>Script Task</i> .....	35
4.1.8 <i>Java Service Task</i> .....	36
4.1.9 <i>WebService Task</i> .....	38
<i>And here is the class:</i> .....	39
4.1.10 <i>Java receive task</i> .....	39
4.2 DESIGNING AND RUNNING A SIMPLE ORCHESTRATION.....	40
4.2.1 <i>WebService Development</i> .....	40





4.2.2	Process model Design.....	42
4.2.3	Form Creation.....	43
4.2.4	XML refinement.....	45
4.2.5	Deploying our Process.....	47
4.2.6	Running the process.....	49
<b>5</b>	<b>EXTENDING THE INFRASTRUCTURE TO RUN CHOREOGRAPHIES.....</b>	<b>50</b>
5.1	EXISTING TOOLS AND WORKAROUND.....	52
5.1.1	Instantiation.....	52
5.1.2	Send messages.....	52
5.1.3	Receive message.....	52
5.2	CASE STUDY.....	53
5.3	FIRST IMPLEMENTATION SAME ENGINE MULTIPLE CLASSES.....	60
5.4	SECOND IMPLEMENTATION DIFFERENT ENGINE MULTIPLE SERVICES.....	62
5.5	FINAL IMPLEMENTATION DIFFERENT ENGINE, SINGLE RECEIVING SERVICES.....	64
5.6	EXTENSION SUMMARY.....	66
	<b>CONCLUSIONS AND FUTURE WORK.....</b>	<b>67</b>
	<b>REFERENCES.....</b>	<b>68</b>

## TABLE LIST

<b>Tables</b>	<b>Page</b>
Table 1 : Database Tables.	15



## FIGURE LIST

Figures	Page
Figure 1 : DataBase Schema .....	16
Figure 2 : Activiti Designer Design perspective.....	17
Figure 3 : Activiti Designer Xml perspective.....	17
Figure 4 : Login Screen.....	20
Figure 5 : My Tasks view .....	22
Figure 6 : Unassigned tasks view.....	23
Figure 7 : Processes page.....	24
Figure 8 : Deployments page .....	25
Figure 9 : start event .....	26
Figure 10 : End event.....	27
Figure 11 : Sequence flow .....	28
Figure 12 : conditional sequence flow .....	28
Figure 13 : conditional sequence flow example .....	29
Figure 14 : Gateways .....	29
Figure 15 : exclusive gateway.....	30
Figure 16 : exclusive gateway example .....	31
Figure 17 : parallel Gateway.....	33
Figure 18 : Parallel Gateway example .....	34
Figure 19 : User Task.....	35
Figure 20 : Script Task.....	36
Figure 21 : Java Service Task .....	37
Figure 22 : WebService Task.....	38
Figure 23 : receive task .....	40
Figure 24 : converter service wsdl representation .....	41
Figure 25 : Converter BPMN 2 model.....	42
Figure 26 : uploading deployment .....	48
Figure 27 : Convert temperature form .....	49
Figure 28 : Convert result form .....	50
Figure 29 : Buyer order process.....	54
Figure 30 : seller process .....	55
Figure 31 : Order Request form .....	56
Figure 32 : make offer form.....	57
Figure 33 : assess offer form.....	58
Figure 34 : Complete order form .....	59



## CODE SNIPPET LIST

Snippets	Page
: login.xml.....	20
: login.ftl.....	21
: start event xml.....	27
: end event.....	27
: Sequence flow.....	28
: conditional sequence flow.....	29
: exclusive gateway xml.....	32
: Parallel Gateway xml.....	34
: User Task Xml.....	35
: Script Task.....	36
: WebService Task Xml.....	39
: WebService call class.....	39
: converter.java file.....	41
: conversion start form.....	43
: conversion result form.....	44
: Unrefined xml converter model.....	45
: refined service task.....	46
: refined user tasks.....	46
: sequence flow refinement.....	47
: get Process instance execution by process id.....	52
: send signal to execution.....	53
: Order Request form.....	56
: make offer form.....	57
: assess offer form.....	58
: Complete order form.....	59
: Delegate class for sending a new order.....	60
: Delegate class for sending the offer.....	61
: delegate class to make Http calls.....	63
: Start Shopper webScript.....	64
: global receiver WebScript.....	66



## **1 INTRODUCTION**

In this thesis we are going to describe the process that was followed for the development of an infrastructure where a user will be able to easily design, develop and run service choreographies. In the first part we are going to introduce and describe the fundamentals of web service technologies. Later we shall go through the existing tools that we used in order to set the base of our infrastructure. As existing tools are not able to develop and run choreographies in the third part we will analyze the extensions developed for these tools by us, so as to augment these tools to be able to fulfill our need, which as we previously mentioned is to run choreographies. These extensions introduce new communication and data exchange mechanisms that are required. Furthermore we shall describe the process that a user must follow in order to design and run a proper choreography using these tools and our extensions. This process is our own approach to the problem and is basically our proposal and demonstration of how a choreography infrastructure should work.

## **2 BACKGROUND KNOWLEDGE/THEORY**

### **2.1 Web services**

The first element we should be acquainted with is web services. Web service is our fundamental piece. It is basically to a web process and application what a class is to a Java application with the significant difference that a web service can exist and be called from wherever in the net.

The W3C[35] defines a "Web service" [7] as a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Services Description Language WSDL[6]). Other systems interact with the Web service in a manner prescribed by its description using SOAP[5] messages, typically conveyed using HTTP[19] with an XML[8] serialization in conjunction with other Web-related standards."

Furthermore we can identify two major classes of Web services, REST-compliant Web services [28], in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations; and arbitrary Web services, in which the service may expose an arbitrary set of operations. The modern trend is towards RESTful services as they lack the complexity using well known and established HTTP protocols [26].



## 2.2 Orchestrations

Although a Web Service may expose many methods, each WSDL (Web Service Description Language file) describes fairly atomic, low-level functions. What a single service does not give us is the rich behavioral detail that describes the role the service plays as part of a larger, more complex collaboration. When these collaborations and collections of activities are designed to accomplish a given business objective, they are known as a business process. A business process may extend across one or more organizations. The description of the sequence of activities that make up a business process is called an orchestration.

## 2.3 Choreographies

Service choreography is a form of service composition in which the interaction protocol between several partner services is defined from a global perspective. The intuition underlying the notion of service choreography can be summarized as follows:

“Dancers dance following a global scenario without a single point of control”

That is, at run-time each participant in service choreography executes its part of it (i.e. its role) according to the behavior of the other participants. Choreography’s role specifies the expected messaging behavior of the participants that will play it in terms of the sequencing and timing of the messages that they can consume and produce.

## 2.4 Orchestrations and Choreographies

The primary difference between orchestration and choreography is executability and control. An orchestration specifies an executable process that involves message exchanges with other systems, such that the message exchange sequences are controlled by the orchestration designer. Choreography specifies a protocol for peer-to-peer interactions, defining, e.g., the legal sequences of messages exchanged with the purpose of guaranteeing interoperability. Such a protocol is not directly executable, as it allows many different realizations (processes that comply with it). A choreography can be realized by writing an orchestration (e.g. in the form of a BPEL process) for each peer involved in it. The orchestration and the choreography distinctions are based on analogies: orchestration refers to the central control (by the conductor) of the behavior of a distributed system (the orchestra consisting of many players), while choreography refers to a distributed system (the dancing team) which operates according to rules (the choreography) but without centralized control [16].

## 2.5 BPMN

Both the concept of orchestrations and choreographies is just theoretical. In order to design and develop one we need an appropriate language. As these architectures were



developed partly having in mind business models and business management people, many languages that aroused were graphical. The most notable one is BPMN.

More specifically Business Process Modeling Notation (BPMN)[12] is a graphical representation for specifying business processes in a business process model.

The Business Process Modeling Notation (BPMN) is a standard for business process modeling, and provides a graphical notation for specifying business processes in a Business Process Diagram (BPD)[27], based on a flowcharting technique very similar to activity diagrams from Unified Modeling Language (UML)[38]. The objective of BPMN is to support business process management for both technical users and business users by providing a notation that is intuitive to business users yet able to represent complex process semantics. The BPMN specification also provides a mapping between the graphics of the notation to the underlying constructs of execution languages, particularly Business Process Execution Language.

The primary goal of BPMN is to provide a standard notation that is readily understandable by all business stakeholders. These business stakeholders include the business analysts who create and refine the processes, the technical developers responsible for implementing the processes, and the business managers who monitor and manage the processes. Consequently, BPMN is intended to serve as common language to bridge the communication gap that frequently occurs between business process design and implementation.

The weaknesses of BPMN could relate to:

- Ambiguity and confusion in sharing BPMN models
- Support for routine work
- Support for knowledge work, and
- Converting BPMN models to executable environments

## **2.6 BPEL**

Unlike BPMN other languages with developers in mind where evolved. The most notable one is BPEL.

Business Process Execution Language (BPEL), short for Web Services Business Process Execution Language (WS-BPEL)[32] is an OASIS[34] standard executable language for specifying actions within business processes with web services. Processes in Business Process Execution Language export and import information by using web service interfaces exclusively. BPEL is an orchestration language, not a choreography language.

WS-BPEL is meant to be used to model the behavior of both Executable and Abstract Processes

Some BPEL features are:

- Facilities to enable sending and receiving messages. A property-based message correlation mechanism



- XML and WSDL typed variables
- An extensible language plug-in model to allow writing expressions and queries in multiple languages: BPEL supports XPath 1.0 by default
- Structured-programming constructs including if-then-elseif-else, while, sequence (to enable executing commands in order) and flow (to enable executing commands in parallel)
- A scoping system to allow the encapsulation of logic with local variables, fault-handlers, compensation-handlers and event-handlers
- Serialized scopes to control concurrent access to variables

## 2.7 BPEL and BPMN

The BPMN specification includes an informal and partial mapping from BPMN to BPEL [17]. A more detailed mapping of BPMN to BPEL has been implemented in a number of tools, including an open-source tool known as BPMN2BPEL. However, the development of these tools has exposed fundamental differences between BPMN and BPEL, which make it very difficult, and in some cases impossible, to generate human-readable BPEL code from BPMN models. Even more difficult is the problem of BPMN-to-BPEL round-trip engineering: generating BPEL code from BPMN diagrams and maintaining the original BPMN model and the generated BPEL code synchronized, in the sense that any modification to one is propagated to the other.

## 2.8 BPMN 2

The vision of BPMN 2.0 [9] is to have one single specification for a new Business Process Model and Notation that defines the notation, metamodel and interchange format. The features include:

Aligning BPMN with the business process definition meta model BPDM to form a single consistent language

Enabling the exchange of business process models and their diagram layouts among process modeling tools to preserve semantic integrity

Expand BPMN to allow model orchestrations and choreographies as stand-alone or integrated models

Support the display and interchange of different perspectives on a model that allow a user to focus on specific concerns

Serialize BPMN and provide XML schemes for model transformation and to extend BPMN towards business modeling and executive decision support.

As a result BPMN 2 is expected to replace and expand the existing solutions that use BPMN and BPEL. Merging the design and executing phase of developing orchestrations and choreographies.



## **3 THE INFRASTRUCTURE**

In this section as mention previously we are going to describe the tools we are going to combine in order to form the infrastructure to fulfill our requirements that is no other that to provide an easy environment where choreographies can be designed and executed. So far there are numerous toolkits that allow a user to develop Service-Oriented Architecture (SOA) Applications. Most of these tools usually have a design environment based on using a notation like BPMN which they later map to BPEL. Furthermore they have an engine that will run the BPEL scripts over a web server and a hosting environment for web services. Having a BPEL as its engine obviously restricts the infrastructure to the restrictions o BPEL. As a result we can only design orchestrations and not Choreographies.

This obviously means that we have to follow a different root. As our core we will use a BPMN 2 engine. Though BPMN 2 is meant to implement choreographies as this area is still very fuzzy all newly developed engines have overlooked them. So we will have to find an open source solution that will allow as customizing it and extending it to our needs. Other than that our infrastructure architecture looks very similar to existing tools let's start describing our components.

### **3.1 THE BPMN 2 ENGINE**

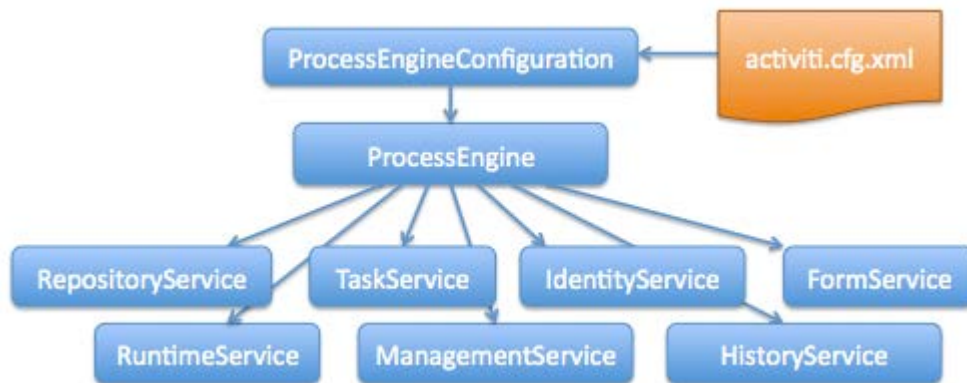
For the engine we will be using the Activity BPMN 2 engine [31]. Activity Engine is a Java process engine that runs BPMN 2 processes. The engine can be incorporates in any java environment as a jar file. In our case the engine is implemented as a web Application and run on Apache Tomcat Server[10].

The Web Application is developed using the Spring Surf technologies[33]. It offers its API through webScripts which are basically restful services that can provide us remote access to the functionality and information of the engine.

**3.1.1 The engines API and its hierarchy can be seen in the following figure.**







Intuitively we can understand that through the ProcessEngine Object we can gain access to all the artifacts that we need.

RepositoryService: Gives access to the process definition.

TaskService: Gives access to the active tasks of an execution.

IdentityService: Give access to user information and authentication.

FormService: Retrieves info about the form files assigned to tasks or start events.

RuntimeService: Gives access to the execution. Accesses variable sends signal and others.

ManagementService deletes or adds new deployments to the engine.

HistoryService gives access to completed tasks and processes.

More specifically the features and functionalities we are interested in are:

- Users: The machine has incorporated a user and group architecture. Our users can be assigned to belong to any number of groups. All the groups can be seen and managed through the DB Table “act\_id\_group”. The groups are separated to two categories.
- Security-Role: These groups are used to define the access level of the user to the functionalities of the engine. A user can only belong to one group of this type. The existing types are User, Manager and System-Administrator. In our site this role gives or restricts access to the deployment page.
- Assignment: The users can belong to any number of assignment type groups. These groups are used to assign a task or a job to possibly any member of a group. This means that once a task, with assignment the “accountancy” group, is reached by an execution in a process all members of the group will be prompt to claim the task, The first one to claim it will be the one to fulfill it.



- Tasks can also be directly assigned to a user by his id or possibly to a number of users by declaring all their ids in the BPMN model.
- Image Representation: The engine has the capability to store an image representation of a model. Later we can use this feature to assist the execution of the models through our management App.
- Forms: Each Human task can be assigned to an HTML form file. This way once a task is reached we can have a Visual representation of the data and information required by the user.
- Execution scope: Every Time a process instance is started an execution is created. This execution can store variables that the tasks can use or declare. If our model has a fork then the execution duplicates itself along with all its variables.

## 3.2 MySql DataBase

For the BPMN 2 engine to run properly a database with the adequate tables must be created. For ease of use a sql dump with the initial entries has been created.

The database schema is described briefly below.

- **ACT\_RE\_\***: 'RE' stands for repository. Tables with this prefix will contain 'static' information such as process definitions and, process resources (images, rules, etc.).
- **ACT\_RU\_\***: 'RE' stands for runtime. These are the runtime tables, which contain the runtime data of process instances, user tasks, variables, jobs, etc. Activiti only stores the runtime data during process instance execution, and removes the records when a process instance ends. This keeps the runtime tables small and fast.
- **ACT\_ID\_\***: 'ID' stands for identity. These tables contain identity information, such as users, groups, etc.
- **ACT\_HI\_\***: 'HI' stands for history. These are the tables that contain historic data, such as past process instances, variables, tasks, etc.
- **ACT\_GE\_\***: general data, which is used in various use cases.

Table Name	Children	Parents	Columns
<a href="#">act_cy_comment</a>	1	1	8
<a href="#">act_cy_config</a>			4
<a href="#">act_cy_conn_config</a>			7
<a href="#">act_cy_link</a>			14



<b>Table Name</b>	<b>Children</b>	<b>Parents</b>	<b>Columns</b>
<a href="#">act_cy_people_link</a>			8
<a href="#">act_cy_tag</a>			5
<a href="#">act_ge_bytearray</a>	2	1	5
<a href="#">act_ge_property</a>			3
<a href="#">act_hi_actinst</a>			11
<a href="#">act_hi_detail</a>			15
<a href="#">act_hi_procinst</a>			10
<a href="#">act_hi_taskinst</a>			13
<a href="#">act_id_group</a>	1		4
<a href="#">act_id_membership</a>		2	2
<a href="#">act_id_user</a>	1		6
<a href="#">act_re_deployment</a>	1		3
<a href="#">act_re_procdef</a>	1		9
<a href="#">act_ru_execution</a>	7	3	11
<a href="#">act_ru_identitylink</a>		1	6
<a href="#">act_ru_job</a>		1	15
<a href="#">act_ru_task</a>	1	3	11
<a href="#">act_ru_variable</a>		3	12

**Table 1 : Database Tables.**

Below we can see a graphical representation of the schema relations.





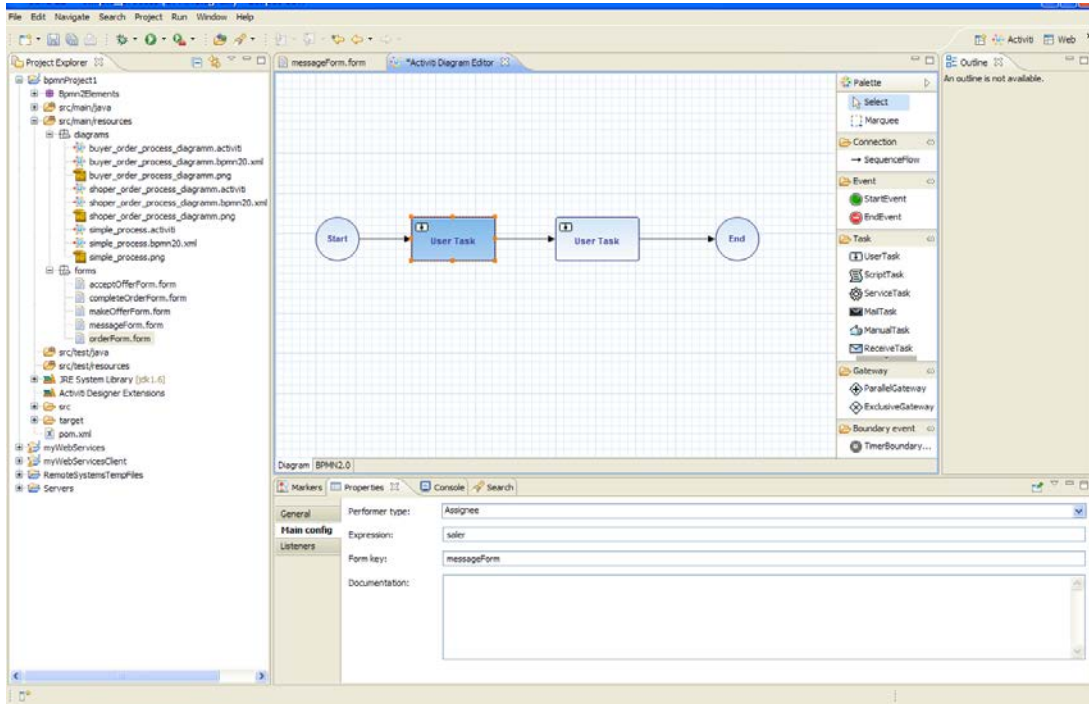


Figure 2 : Activiti Designer Design perspective.

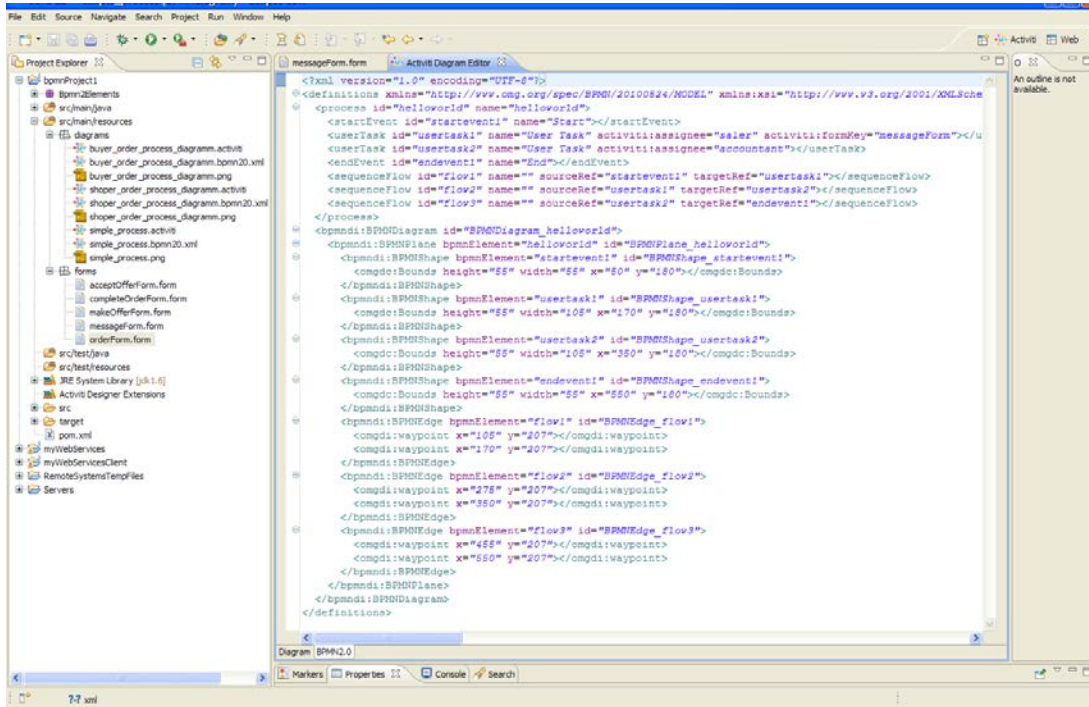


Figure 3 : Activiti Designer Xml perspective.



### 3.4 Management Web Application

For The execution and management of our models it was required to create a Web Application that would offer a visual representation of the information's required to run and observe our processes and of course give us the functionality to interact with them. The detailed functionality of the application will be described later along with an explanatory example. Here we will focus on the framework and the techniques used for its development.

For the development of the web app we used the Spring Surf framework. Below we will describe the main configuration files and artifacts of a Spring surf project.

1. MYWEBAPP\WEB-INF\web.xml  
Defines a "UrlRewriteFilter" filter which Enables clean URLs with JSP views e.g. enabling url /welcome instead of /page/welcome.  
Spring MVC Dispatcher Servlet is also defined and pointed to /WEB-INF/config/web-application-config.xml for its context configuration.
2. MYWEBAPP\WEB-INF\urlrewrite.xml  
This configuration file provides UrlRewriteFilter filter with a set of Surf related inbound and outbound url rewrite rules.
3. MYWEBAPP\WEB-INF\surf.xml  
It defines Surf specific configurations for runtime and mode. As default, it uses webapp runtime and development mode. (TODO: more details on the options)
4. MYWEBAPP\WEB-INF\config\web-application-config.xml  
As defined in web.xml, this file provides context configurations for the Spring MVC Dispatcher Servlet. It first imports required infrastructure imports from MYWEBAPP\WEB-INF\config\surf-config.xml and then defines a list of required interceptors for the default Spring MVC annotation handler. Rest of the configurations are for interoperability with Spring annotated controllers and simple controllers. It also configures the default Spring multipart resolver for file uploading.
5. MYWEBAPP\WEB-INF\config\surf-config.xml  
This configuration file defines context locations for both Surf Web Scripts Framework and Surf Framework. It also sets up to be auto-resolved to url based views.
6. MYWEBAPP\css\sample.css  
This is the style sheet of the Quick Start Sample Site.
7. MYWEBAPP\images\\*  
Image files for the Quick Start Sample Site.
8. MYWEBAPP\WEB-INF\chrome\\*  
Chrome describes the border elements around a region or a component. These border elements may include styling elements like shadows, or they





may introduce drag and drop capabilities into the page. They may also introduce user-functionality like buttons for popping up component settings (as you frequently see in portals).

9. MYWEBAPP\WEB-INF\pages\\*

Our main Pages defined in XML format. Surf only requires page configuration XMLs to be placed under MYWEBAPP\WEB-INF. However for best practice we place them under MYWEBAPP\WEB-INF\pages example MYWEBAPP\WEB-INF\pages\login.xml.

A page is a navigable page in our web application. It may have associations to other pages and multiple formats keyed from it to templates. A page is a top-level object from which you can traverse either additional navigation or rendering paths.

10. MYWEBAPP\WEB-INF\templates\\*

The template folder. A Template Instance is an instance of a template type, which can be of a Freemarker type (templateName.ftl)[20] JSP or WebScript. A Surf page object has a required field for template instance therefore the Surf dispatcher knows which template to use to render view for the page. While Page xml configuration files describe the relations of a page the templates determine their appearance.

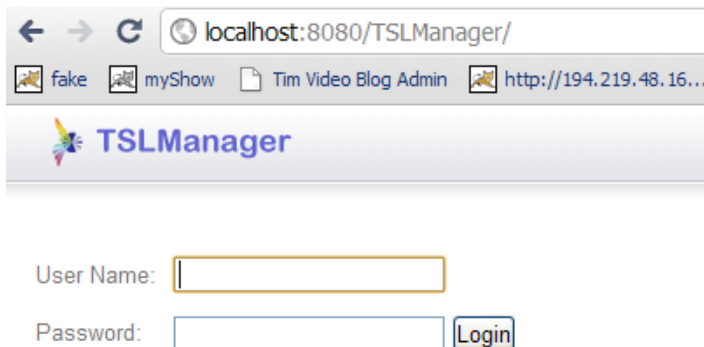
11. MYWEBAPP\WEB-INF\webscripts\\*

This is the place for storing WebScripts that are generating components such as header, footer, navigation etc. Webscripts are like restful services and separate to view from the control.

A component is an instance of a component type that has been "bounded" into a region or a slot. It represents a binding along with the instance-specific state for that component instance. The Surf framework supports three types of scopes for the region/component binding, global scope, template scope and page scope. The Global Scope is for the component that is same across the site such as header and footer. The Template Scope is for the component that is same across the template such as a Text Block component which shows the same text for any page using this template. The page scope is for any page specific component.



For Better understanding we shall show how the simple page of login is represented.



**Figure 4 : Login Screen**

```
<page>
  <title>Login</title>
  <title-id>page.login.title</title-id>
  <template-instance>login</template-instance>
  <authentication>none</authentication>
</page>
```

#### **Code Snippet 1 : login.xml**

In login.xml we can see its title its id the template instance that will be used to represent it and its authentication set to none, meaning that anyone can access this page (No login required prior to this page).

```
<#include "activiti.template.lib.ftl" />
<#assign successUrl=(url.args["url"]!url.url)?html />
<#assign failureUrl="/login?error=true" />
<@templateHeader/>
<@templateBody>
  <div id="header">
    <div class="activiti-component">
      <div class="application-info">
        
```





```
</div>
</div>
</div>
<div id="content">
  <div id="login">
    <form accept-charset="UTF-8" method="post" action="{url.context}/dologin">
      <#if !successUrl?contains(failureUrl)>
        <input name="success" type="hidden" value="{successUrl}" />
      </#if>
      <input name="failure" type="hidden"
value="{url.context}{failureUrl}&url={successUrl?url}" />
      <#if url.args["error"]??>
        <div class="section">
          <div id="login-error" class="status-error"></div>
        </div>
      <#else>
        <div class="section">
          <div id="login-browser-warning" class="status-error"></div>
        </div>
      </#if>
      <div class="section">
        <label for="username">User Name:</label>
        <input id="username" name="username" type="text" />
        <br/>
      </div>
      <div class="section">
        <label for="password">Password:</label>
        <input id="password" name="password" type="password" />
        <input id="submit" type="submit" value="Login"/>
        <br/>
      </div>
    </form>
  </div>
</div>
</@>
<@templateFooter/>
```

**Code Snippet 2 : login.ftl**

In This freemarker file we can see the main form and the error control depending on the incoming url.



### 3.4.1 Architecture and functionalities

Our Site is basically consisted of two main pages two child pages and several components.

Our first main page was described before and is the **login** screen of our system. Once we are logged in we can access the applications main page **start**. The **start** page has two child pages and as defined by its template a header and footer. Since the other two pages are children of this they share the same header and footer. These three pages have the following function.

1. **Start/tasks** Once in this page we use our task component to represent the active tasks for these users. The Tasks are separated to personal, unassigned and by owner group. Since a user can belong two several groups the tasks are categorized. Unassigned tasks appear with a claim selection. Once claimed the task is now personal and appears only to this user from now one with the option to complete the task.

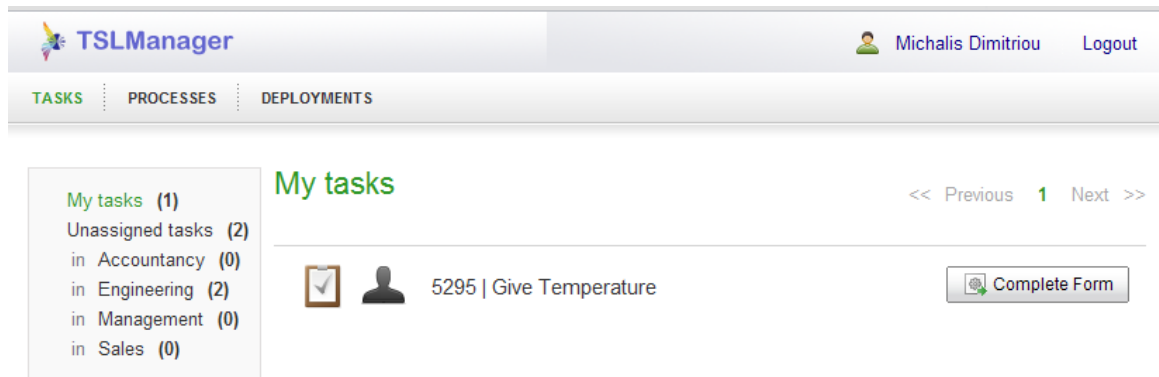


Figure 5 : My Tasks view



The screenshot shows the TSLManager web application interface. At the top, there is a navigation bar with the TSLManager logo and the user name 'Michalis Dimitriou' with a 'Logout' link. Below the navigation bar, there are three tabs: 'TASKS', 'PROCESSES', and 'DEPLOYMENTS'. The 'TASKS' tab is active. On the left side, there is a sidebar with a list of task counts: 'My tasks (1)', 'Unassigned tasks (2)', 'in Accountancy (0)', 'in Engineering (2)', 'in Management (0)', and 'in Sales (0)'. The main content area is titled 'Unassigned tasks in group Engineering' and shows two tasks. Each task has a checkbox, a user icon, a task ID, a description, and a 'Claim' button. The first task is '5337 | Investigate hardware' and the second is '5340 | Investigate software'. There are navigation arrows at the top right of the task list: '<< Previous 1 Next >>'. The 'Claim' buttons are located to the right of each task entry.

Figure 6 : Unassigned tasks view

2. **Processes** The first child page is **PROCESSES**. Here we can see all the deployed processes. If a graphical representation of the model has been given we have the option to view it. Otherwise we have only the option the initiate the process. If the start event of the process was assigned with a form then the option to fill that form is presented. At this point anyone can start any process thaw afterwards only assigned users can interact with its tasks.



The screenshot shows the TSLManager web application interface. At the top, there is a navigation bar with 'TASKS', 'PROCESSES', and 'DEPLOYMENTS' tabs. The 'PROCESSES' tab is active. The main content area is titled 'Processes' and displays a list of processes. Each process entry includes a name, a version indicator (v1), and two buttons: 'Show Diagram' and 'Start Instance'. The processes listed are: Expense process, Timer escalation example, Fix system failure, Write bi-monthly financial report, Review sales lead, Single candidate group example, Schedule meeting reminder, TempConverter, and Vacation request. The 'Vacation request' process has an additional 'Show Start Form' button. The user 'Saler Saler' is logged in, and the page number '1' is visible in the pagination controls. The footer contains the copyright notice: © 2011 tsl.gr. All rights reserved.

Figure 7 : Processes page

3. **Deployments** Finally the second Children is **DEPLOYMENTS**. In this Page a user has access only if he belongs to the administrator group. Here someone can delete or upload new deployments. Every deployment can include several processes and its accompanying files. The way to deploy a new process will be described later.



TSLManager Michalis Dimitriou Logout

TASKS PROCESSES DEPLOYMENTS

## Deployments

<< Previous 1 Next >>

Select	Id ▲	Name	Time
<input type="checkbox"/>	5267	converter.bar	Fri 15 Apr 2011 01:32:32
<input type="checkbox"/>	5296	activiti-engine-examples.bar	Fri 15 Apr 2011 04:39:46

Delete Delete Cascade Upload

Figure 8 : Deployments page

### 3.5 Summary

So far we have combined a set of tools and frameworks that allow as:

1. Run simple web service.
2. Design orchestrations based on BPMN 2.
3. Run those BPMN 2 scripts.
4. A web environment where we can manage users, services, processes, and monitor their flow.

Basically what a simple user will see is a web based process management system. In this system we have users that can initiate processes and provide data to the system when asked (human task), furthermore we have automated tasks running by the system (web services). Finally the flow of the execution of those tasks is directed by the models designed in BPMN 2.



What we lack is extending orchestration to choreographies. To achieve that we need to find a way for multiple processes (orchestrations) existing in the same or different environment to communicate with each other and basically self coordinate their operations which will ultimately form a choreography.

## **4 THE FRAMEWORK IN PRACTICE**

In order to understand the concept that led us to the development of our extensions we must understand how everything works through examples.

### **4.1 Learning the BPMN 2.0 constructs**

Before we start our example we must familiarize a little with the BPMN 2.0 artifacts their graphical and xml representation.

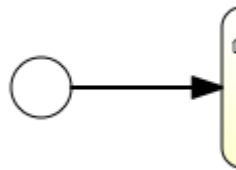
#### **4.1.1 None start event**

##### **Description**

A 'none' start event means that there is no trigger for starting the process instance. This means that the engine cannot anticipate when the process instance must be started. As a result we must start the instance programmatically. A good practice is through a web service which we can call from an interface or can be called by another process.

##### **Graphical notation**

A “none” start event is graphically represented as a circle with no inner icon (i.e. no trigger type).



**Figure 9 : start event**

##### **XML representation**



The XML representation of a none start event is the normal start event declaration, without any sub-element (other start event types all have a sub-element declaring the type).

```
<startEvent id="start" name="my start event" />
```

**Code Snippet 3 : start event xml**

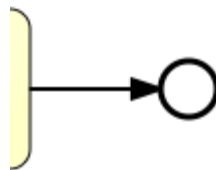
## 4.1.2 None end event

### Description

A 'none' end event means that there is no *result* thrown when the end event is reached. As such, the engine will not do anything besides ending the current path of execution. Notice that the end event doesn't end the process but the current path.

### Graphical notation

A "none" end event is graphically represented as a circle with a thick border with no inner icon (no result type).



**Figure 10 : End event**

### XML representation

The XML representation of a none end event is the normal end event declaration, without any sub-element (other end event types all have a sub-element declaring the type).

```
<endEvent id="end" name="my end event" />
```

**Code Snippet 4 : end event**

## 4.1.3 Sequence flow

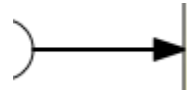
### Description

A sequence flow is the connector between two elements of a process. After an element is visited during process execution path, all outgoing sequence flow will be followed. This means that the default nature of BPMN 2.0 is to be parallel: two outgoing sequence flow will create two separate, parallel paths of execution.

### Graphical notation



A sequence flow is graphically represented as an arrow going from the source element towards the target element. The arrow always points towards the target.



**Figure 11 : Sequence flow**

### XML representation

Sequence flow need to have a process-unique **id**, and a reference to an existing **source** and **target** element.

```
<sequenceFlow id="flow1" sourceRef="theStart" targetRef="theTask" />
```

**Code Snippet 5 : Sequence flow**

## 4.1.4 Conditional sequence flow

### Description

A sequence flow can have a condition attached to it. When an activity is finished, conditions on the outgoing sequence flow are evaluated. When the condition is *true*, that outgoing sequence flow is selected. When multiple sequence flows are true, multiple *paths* will be generated and the process will be continued in a parallel way.

### Graphical notation

A conditional sequence flow is graphically represented as a regular sequence flow, with a small diamond at the beginning. The condition expression is shown next to the sequence flow.



**Figure 12 : conditional sequence flow**

### XML representation

A conditional sequence flow is represented in XML as a regular sequence flow, containing a **condition Expression** sub-element. Note that for the moment only *tFormalExpressions* are supported, Omitting the *xsi:type=""* definition will simply default to this only supported type of expressions.





```
<sequenceFlow id="flow" sourceRef="theStart" targetRef="theTask">  
  <conditionExpression xsi:type="tFormalExpression">  
    <![CDATA[ $\{order.price > 100 \ \&\& \ order.price < 250\}$ ]]>  
  </conditionExpression>  
</sequenceFlow>
```

Code Snippet 6 : conditional sequence flow

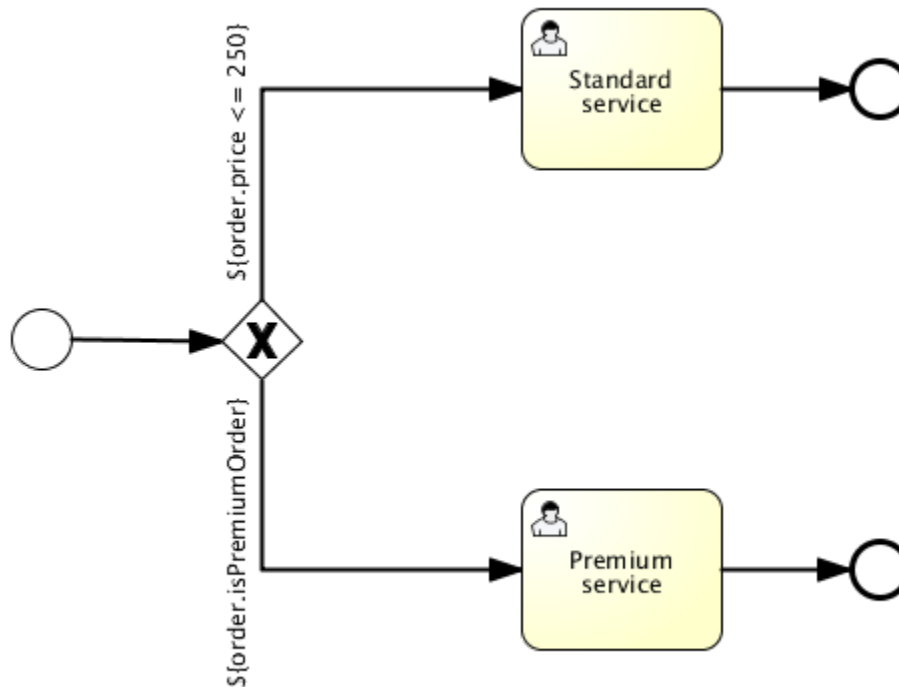


Figure 13 : conditional sequence flow example

#### 4.1.5 Gateways

A gateway is used to control the path of execution. A gateway is graphically represented as a diamond shape, with an icon inside. The icon shows the type of gateway.

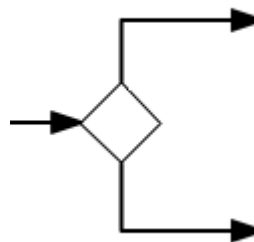


Figure 14 : Gateways



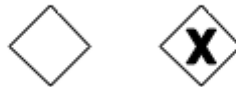
#### 4.1.5.1 Exclusive gateway

##### Description

An exclusive gateway (*XOR gateway*), is used to model a single **decision** in the process. When the execution arrives at this gateway, all outgoing sequence flows are evaluated in the order in which they are defined. The first sequence flow whose condition evaluates to true is selected for continuing the process.

##### Graphical notation

An exclusive gateway is graphically represented as a typical gateway (i.e. a diamond shape) with an 'X' icon inside, referring to the *XOR* semantics. Note that a gateway without an icon inside defaults to an exclusive gateway.

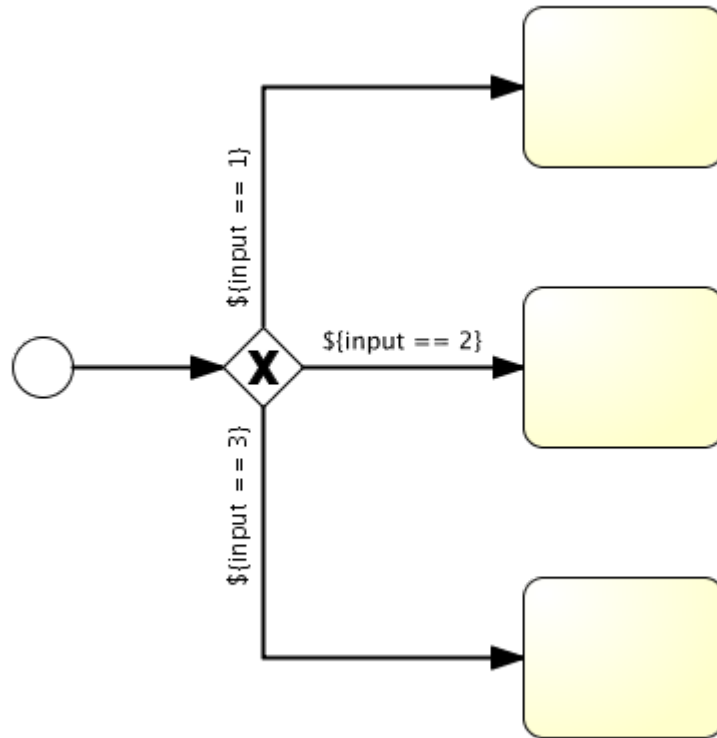


**Figure 15 : exclusive gateway**

##### XML representation

The XML representation of an exclusive gateway is straight-forward: one line defining the gateway and condition expressions defined on the outgoing sequence flow. Take for example the following model:





**Figure 16 : exclusive gateway example**



This is represented in XML as follows:

```
<exclusiveGateway id="exclusiveGw" name="Exclusive Gateway" />

<sequenceFlow id="flow2" sourceRef="exclusiveGw" targetRef="theTask1">
  <conditionExpression xsi:type="tFormalExpression">${input ==
1}</conditionExpression>
</sequenceFlow>

<sequenceFlow id="flow3" sourceRef="exclusiveGw" targetRef="theTask2">
  <conditionExpression xsi:type="tFormalExpression">${input ==
2}</conditionExpression>
</sequenceFlow>

<sequenceFlow id="flow4" sourceRef="exclusiveGw" targetRef="theTask3">
  <conditionExpression xsi:type="tFormalExpression">${input ==
3}</conditionExpression>
</sequenceFlow>
```

**Code Snippet 7 : exclusive gateway xml**

#### 4.1.5.2 Parallel Gateway

##### Description

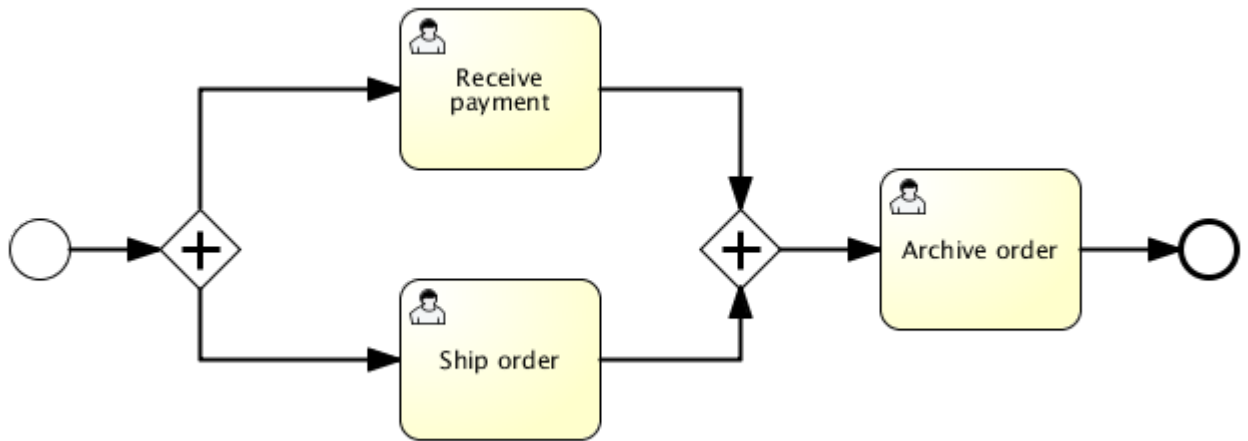
Gateways can also be used to model concurrent paths in a process. This is the **Parallel Gateway**, which allows forking into multiple paths or joining multiple incoming paths. Basically The Parallel Gateway does nothing as it leaves all the path selection to the conditions in the sequence flows. Based on the incoming and outgoing sequence flow a parallel Gateway can have any of the following functions or both (multiple incoming and outgoing flows):

4. fork: all outgoing sequence flow are followed in parallel, creating one concurrent execution for each sequence flow.
5. join: all concurrent executions arriving at the parallel gateway wait in the gateway until an execution has arrived for each of the incoming sequence flow. Then the process continues past the joining gateway.

##### Graphical Notation

A parallel gateway is visualized as a gateway (diamond shape) with the 'plus' symbol inside, referring to the 'AND' semantics.





**Figure 17 : parallel Gateway**

### XML representation

Defining a parallel gateway needs one line of XML:

```
<parallelGateway id="myParallelGateway" />
```

The actual behavior (fork, join or both), is defined by the sequence flow connected to the parallel gateway.

For example, the model above comes down to the following XML:

```
<startEvent id="theStart" />
<sequenceFlow id="flow1" sourceRef="theStart" targetRef="fork" />

<parallelGateway id="fork" />
<sequenceFlow sourceRef="fork" targetRef="receivePayment" />
<sequenceFlow sourceRef="fork" targetRef="shipOrder" />

<userTask id="receivePayment" name="Receive Payment" />
<sequenceFlow sourceRef="receivePayment" targetRef="join" />

<userTask id="shipOrder" name="Ship Order" />
<sequenceFlow sourceRef="shipOrder" targetRef="join" />

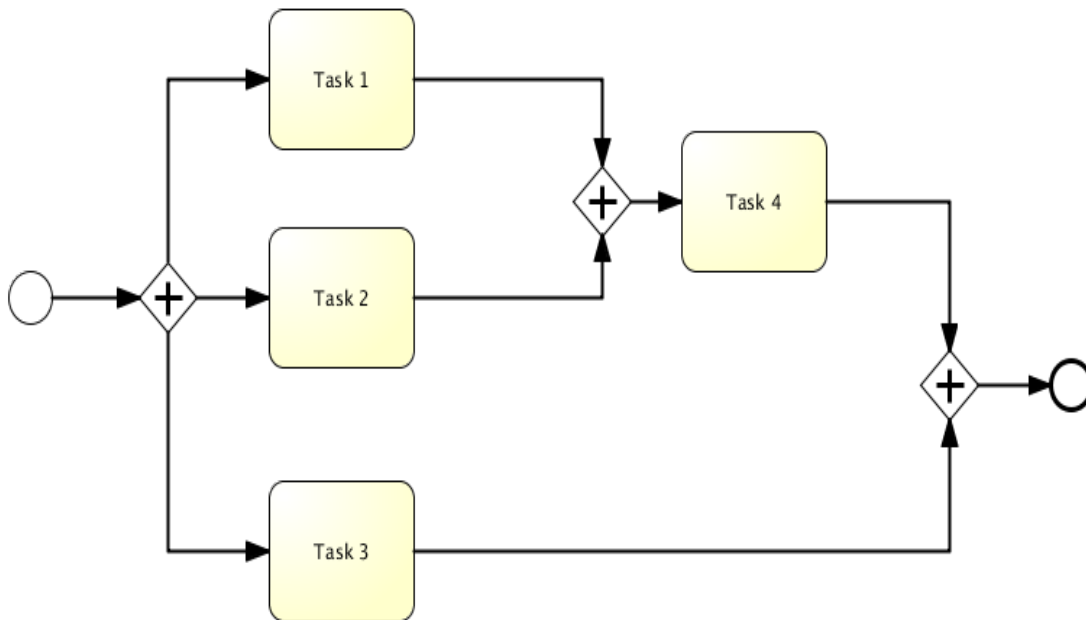
<parallelGateway id="join" />
<sequenceFlow sourceRef="join" targetRef="archiveOrder" />

<userTask id="archiveOrder" name="Archive Order" />
<sequenceFlow sourceRef="archiveOrder" targetRef="theEnd" />
```



```
<endEvent id="theEnd" />
```

**Code Snippet 8 : Parallel Gateway xml**



**Figure 18 : Parallel Gateway example**

#### 4.1.6 User task

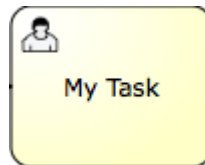
##### Description

A 'user task' is used to model task that needs to be performed by a human.

##### Graphical notation

A user task is graphically represented as a typical task (rounded rectangle), with a small user icon in the left upper corner.





**Figure 19 : User Task**

### XML representation

A user task is defined in XML as follows. The *id* attribute is required, the *name* attribute is optional.

```
<userTask id="theTask" name="Important task" />
```

A user task can also have a description under the <documentation> child tag.

```
<userTask id="theTask" name="Schedule meeting" >  
  <documentation>  
    Schedule an engineering meeting for next week with the new hire.  
  </documentation>
```

### Code Snippet 9 : User Task Xml

#### Activiti extensions for task assignment

- **Assignee attribute:** this custom extension allows to directly assign a user task to a given user.

```
<userTask id="theTask" name="my task" activiti:assignee="mdimitr" />
```

- **Candidate Users attribute:** this custom extension allows us to make a user a candidate for a task.

```
<userTask id="theTask" name="my task" activiti:candidateUsers="mdimitr,  
saler" />
```

Candidate means that this user will later have the choice to claim this task or let it to be claimed by another.

- **Candidate Groups attribute:** this custom extension allows making a group as candidate for a task.

```
<userTask id="theTask" name="my task"  
activiti:candidateGroups="management, accountancy" />
```

As users belong to groups then we can assign all users in a group to be candidate users for this task.

## 4.1.7 Script Task

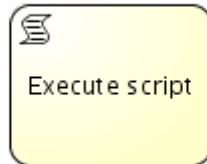
### Description



A script task is an automatic activity. When a process execution arrives at the script task, the corresponding script is executed.

### **Graphical Notation**

A script task is graphically represented as a typical task (rounded rectangle), with a small 'script' icon in the top-left corner of the rectangle.



**Figure 20 : Script Task**

### **XML representation**

A script task is graphically represented by specifying the **script** and the **scriptFormat**.

```
<scriptTask id="theScriptTask" name="Execute script" scriptFormat="groovy">
  <script>
    sum = 0
    for ( i in inputArray ) {
      sum += i
    }
  </script>
</scriptTask>
```

### **Code Snippet 10 : Script Task**

The groovy jar is the default script library for the engine.

## **4.1.8 Java Service Task**

### **Description**

A Java service task is used to invoke an external Java class.

### **Graphical Notation**

A service task is visualized as a rounded rectangle with a small gear icon in the top-left corner.







**Figure 21 : Java Service Task**

### XML representation

There are 4 ways of declaring how to invoke Java logic:

- Specifying a class that implements JavaDelegate or ActivityBehavior
- Evaluating an expression that resolves to a delegation object
- Invoking a method expression
- Evaluating a value expression

To specify a class that is called during process execution, the fully qualified classname needs to be provided by the '**activiti:class**' attribute.

```
<serviceTask id="javaService"  
  name="My Java Service Task"  
  activiti:class="org.activiti.MyJavaDelegate" />
```

It is also possible to use an expression that resolves to an object. This object must follow the same rules as objects that are created when the **activiti:class** attribute is used (see [further](#)).

```
<serviceTask id="serviceTask"  
activiti:delegateExpression="{delegateExpressionBean}" />
```

Here, the **delegateExpressionBean** is a bean that implements the JavaDelegate interface, defined in for example the Spring container.

To specify a method expression that should be evaluated, use attribute **activiti:expression**.

```
<serviceTask id="javaService"  
  name="My Java Service Task"  
  activiti:expression="{printer.printMessage(execution, myVar)}" />
```

Method **printMessage** will be called on the named object called **printer**. This object is by default initialized in every process execution. The first parameter passed is the **DelegateExecution**, which is available in the expression context by default available as **execution**. The second parameter passed, is the value of the variable with name **myVar** in the current execution.



### Service task results

The return value of a service execution (for service task using expression only) can be assigned to an already existing or to a new process variable by specifying the process variable name as a literal value for the '*activiti:resultVariable*' attribute of a service task definition.

```
<serviceTask id="aMethodExpressionServiceTask"  
  activiti:expression="#{myService.doSomething()}"  
  activiti:resultVariable="myVar" />
```

## 4.1.9 WebService Task

### Description

A WebService task is used to synchronously invoke an external web service.

### Graphical Notation

A WebService task is graphically represented the same as a Java service task.

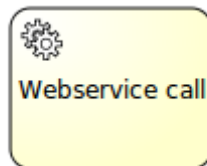


Figure 22 : WebService Task

For less complexity we have created our own class to call webservices so we can instead use the java service task.

```
<serviceTask id="Call_WS" name="Call WS"  
  activiti:class="gr.tsl.delegate.WsDelegate" >  
  <extensionElements>  
    <activiti:field name="wsdl"  
  expression="http://localhost:8080/VacationService?wsdl" />  
    <activiti:field name="operation" expression="saveVacationApproval" />  
    <activiti:field name="parameters" expression="{user}, {days}" />  
    <activiti:field name="returnValue" expression="myReturn" />  
  </extensionElements>
```



```
</serviceTask>
```

### **Code Snippet 11 : WebService Task Xml**

And here is the class:

```
public class WsDelegate implements org.activiti.engine.delegate.JavaDelegate {  
    private Expression wsdl;  
    private Expression operation;  
    private Expression parameters;  
    private Expression returnValue;  
    public void execute(DelegateExecution execution){  
        String wsdlString = (String)wsdl.getValue(execution);  
        JaxWsDynamicClientFactory dcf = JaxWsDynamicClientFactory.newInstance();  
        Client client = dcf.createClient(wsdlString);  
        ArrayList paramStrings = new ArrayList();  
        if (parameters!=null) {  
            StringTokenizer st = new StringTokenizer( (String)parameters.getValue(execution), ",");  
            while (st.hasMoreTokens()) {  
                paramStrings.add(st.nextToken().trim());  
            }  
        }  
        Object response = client.invoke((String)operation.getValue(execution), paramStrings.toArray(new  
Object[0]));  
        if (returnValue!=null) {  
            String returnVariableName = (String) returnValue.getValue(execution);  
            execution.setVariable(returnVariableName, response);  
        }  
    }  
}
```

### **Code Snippet 12 : WebService call class**

#### **4.1.10 Java receive task**

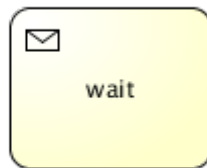
##### **Description**



A receive task is a simple task that waits for the arrival of a certain message. Currently, we have only implemented Java semantics for this task. When process execution arrives at a receive task, the process state is committed to the persistence store. This means that the process will stay in this wait state, until a specific message is received by the engine, which triggers the continuation of the process past the receive task.

### **Graphical notation**

A receive task is visualized as a task (rounded rectangle) with a message icon in the top left corner. The message is white (a black message icon would have send semantics)



**Figure 23 : receive task**

### **XML representation**

```
<receiveTask id="waitState" name="wait" />
```

To continue a process instance that is currently waiting at such a receive task, we must send a signal programmatically for example through a web service we have created for this job.

## **4.2 Designing and running a simple Orchestration**

Our case study will be a simple orchestration involving both automated Tasks (WebServices) and human related tasks. We are going to design a process representing a unit convertor between Celsius and Fahrenheit.

### **4.2.1 WebService Development**

First of all we must implement our WebService. We are going to use eclipse and Apache Axis. To start we implement a simple java bean with the logic we require. We end up with the following Converter.java file.

```
public class Converter
{
    public String celsiusToFahrenheit ( String celsius )
    {
```



```
String tmp=String.valueOf((Float.parseFloat(celsius) * 9 / 5) + 32);
System.out.println(celsius+" C == "+tmp+" F");
return tmp;
}

public String fahrenheitToCelsius ( String fahrenheit )
{
String tmp=String.valueOf((Float.parseFloat(fahrenheit) - 32) * 5 / 9);
System.out.println(fahrenheit+" F == "+tmp+" C");
return tmp;
}
}
```

Code Snippet 13 : converter.java file

Afterwards using the Axis 2 plug-in for Eclipse and Web Services we create a bottoms up java bean WebService and Publish it and its WSDL file on our Server. Here is a graphical representation of our WSDL.

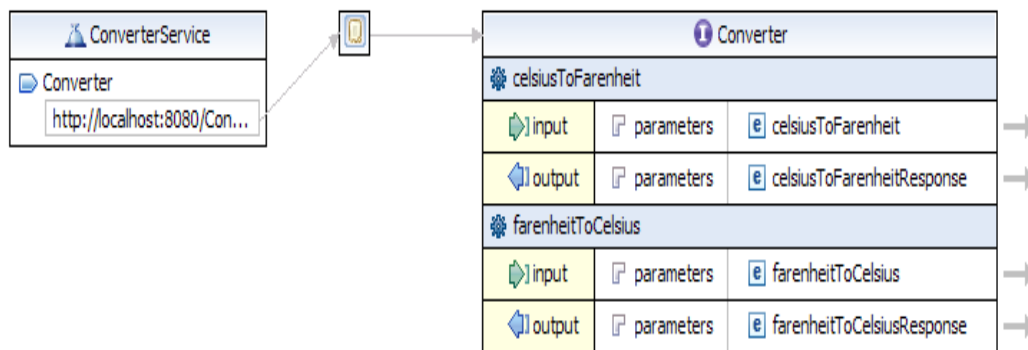


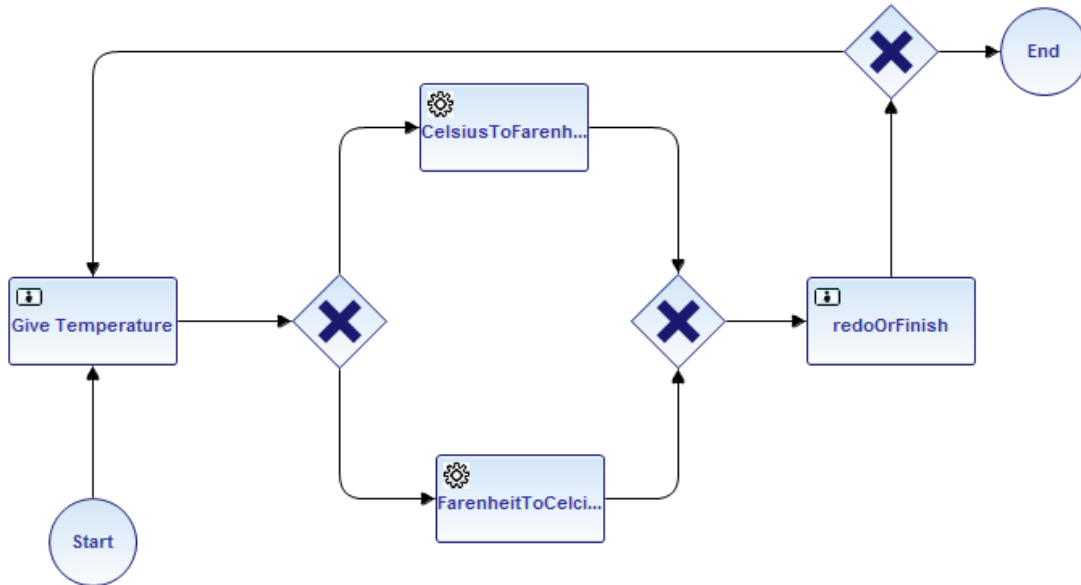
Figure 24 : converter service wsdL representation

As mentioned before in order to call this Web Service we created our own WsDelegate.java class. This Class will make the call and will set the returned value as a Variable in our Execution.



## 4.2.2 Process model Design

Using the Eclipse Activiti Designer we will create the skeleton of our model. Although designer tools can make our life sometimes easier it's usually right to make any refinements directly to the Code. After inserting and connecting all our artifacts we will have something like the following figure.



**Figure 25 : Converter BPMN 2 model**

As can be seen by the Model at the beginning our process a user will be requested to select the type of conversion he wishes to make. Afterward based on his selection the right web service call will be dispatched. Once that is done the user can see the result and choose whether to repeat the process.

After designing our model we will end up with 2 files:

1. ***unit\_converter.bpmn20.xml:***  
This is the xml file of our process.
2. ***unit\_converter.png:***  
This is the graphical representation of the model in png form.



### 4.2.3 Form Creation

For our process to run, it is required that we produce some form files. These files will define how the information for our user will be viewed and the interface for him to make selections and input data. Form file can be assigned to any Start Event and user Task. In our model we shall assign two of such files to our two user Tasks respectively.

For the first user task we require a form that will allow him to choose if he wants to convert from Celsius to Fahrenheit or vice versa and a field to input the value to be converted.

```
<h1>Converter</h1>
<p> Which conversion do you want to make </p>
<table>
<tr>
<td>
<select name="toUnit">
<option value="far">Celsius To Farenheit</option>
<option value="cel">Farenheit To Celsius</option>
</select>
<input type="hidden" name="toUnit_type" value="String" />
</td>
</tr>
<tr>
<td>
<label>
<input type="text" name="temp" value="0" />
<input type="hidden" name="temp_required" value="true" />
</label>
</td>
</tr>
</table>
```

#### Code Snippet 14 : conversion start form

After this form is submitted two new variables will appear in our execution. The Variable “toUnit” has stored the type of conversion the user wants to make and we can use it to direct the process flow. The variable “temp” will hold the temperature we’ll pass to our Webservice for conversion.

For our second user task we require a Form that will portray the results of our conversion and give the selection to finish or redo the process.



```
<h1>ConverterResult</h1>
<table>
<tr>
<td>
  ${temp} ${toUnit=="far"? "Fahrenheit": "Celsius"} is converted to ${newTemp}
  ${toUnit=="cel"? "Fahrenheit": "Celsius"}
</td>
</tr>
<tr>
<td>
  <select name="redo">
<option value="true">Yes</option>
<option value="false">No</option>
</select>
<input type="hidden" name="redo_type" value="Boolean" />
</td>
</tr>
</table>
```

### **Code Snippet 15 : conversion result form**

In this form we can also see how we can run java code through our form or inside our models xml file. Whatever is inside the `${...}` is basically java code that runs inside the execution scope much like jsp. Running inside the execution scope of the process means we have access to its variable and in this case we alter our representation based on the existing values of “toUnit”, “temp” and “newTemp” (newTemp being the name of the variable we will later assign the result of the WebService call).





#### 4.2.4 XML refinement

The current State of the XML File is as follows:

```
<process id="tempConverter" name="TempConverter">
  <startEvent id="startevent1" name="Start"></startEvent>
  <serviceTask id="servicetask1" name="CelsiusToFarenheit" >
  </serviceTask>
  <serviceTask id="servicetask2" name="FarenheitToCelcius">
  </serviceTask>
  <exclusiveGateway id="exclusivegateway1" name="Exclusive Gateway">
    </exclusiveGateway>
    <exclusiveGateway id="exclusivegateway3" name="Exclusive Gateway">
    </exclusiveGateway>
    <userTask id="usertask2" name="Give Temperature">
    </userTask>
  <userTask id="usertask1" name="redoOrFinish">
    </userTask>
  <sequenceFlow id="flow4" name="" sourceRef="servicetask2" targetRef="exclusivegateway3">
  </sequenceFlow>
  <sequenceFlow id="flow6" name="" sourceRef="servicetask1" targetRef="exclusivegateway3">
  </sequenceFlow>
  <sequenceFlow id="flow22" name="" sourceRef="exclusivegateway3" targetRef="usertask1">
  </sequenceFlow>
  <sequenceFlow id="flow7" name="" sourceRef="startevent1" targetRef="usertask2">
  </sequenceFlow>
  <exclusiveGateway id="exclusivegateway2" name="Exclusive Gateway">
  </exclusiveGateway>
  <sequenceFlow id="flow14" name="" sourceRef="usertask1" targetRef="exclusivegateway2">
  </sequenceFlow>
  <sequenceFlow id="flow15" name="" sourceRef="exclusivegateway2" targetRef="usertask2">
  </sequenceFlow>
  <endEvent id="endevent1" name="End"></endEvent>
  <sequenceFlow id="flow18" name="" sourceRef="usertask2" targetRef="exclusivegateway1">
  </sequenceFlow>
  <sequenceFlow id="flow19" name="" sourceRef="exclusivegateway1" targetRef="servicetask1">
  </sequenceFlow>
  <sequenceFlow id="flow20" name="" sourceRef="exclusivegateway1" targetRef="servicetask2">
  </sequenceFlow>
  <sequenceFlow id="flow21" name="" sourceRef="exclusivegateway2" targetRef="endevent1">
  </sequenceFlow>
</process>
```

#### Code Snippet 16 : Unrefined xml converter model

As we can easily notice there is nothing more than our tasks gateways and flows that connect them. What we need to do is add conditions to the flows if required, set Assignees and form to the user tasks and define the class and its fields in the service task for the web Service calls.



#### 4.2.4.1 Refining the service tasks

To make the Service Tasks to function properly first of all we must assign them a class. In this case we use our *gr.tsl.delegate.WsDelegate* class that we use to make WebService calls.

The required fields for this class is the WSDL's URI, the operation of the service we will invoke the input variables and the result variable.

```
<serviceTask id="servicetask1" name="CelsiusToFahrenheit"
  activiti:class="gr.tsl.delegate.WsDelegate">
  <extensionElements>
    <activiti:field name="wsdl">
<activiti:string>http://localhost:8080/ConverterProj/wsdl/Converter.wsdl
</activiti:string>
    </activiti:field>
    <activiti:field name="operation">
      <activiti:string>celsiusToFahrenheit</activiti:string>
    </activiti:field>
    <activiti:field name="parameters">
      <activiti:expression>${temp}</activiti:expression>
    </activiti:field>
    <activiti:field name="returnValue">
      <activiti:string>newTemp</activiti:string>
    </activiti:field>
  </extensionElements>
</serviceTask>
```

#### Code Snippet 17 : refined service task

#### 4.2.4.2 Refining the user Tasks

As far as the user tasks are considered, we need to define an owner and a form. In our case we assign the two forms we created before respectively and set as owner the user "mdimitr".

```
<userTask id="usertask2" name="Give Temperature" activiti:assignee="mdimitr"
  activiti:formKey="converterForm.form"></userTask>
  <userTask id="usertask1" name="redoOrFinish" activiti:assignee="mdimitr"
  activiti:formKey="redoForm.form"></userTask>
```

#### Code Snippet 18 : refined user tasks



#### 4.2.4.3 Refining the sequence flows

Considering the sequence flows we are interested only in those that we wish to control the flow or more practically the ones which we wish to be active under condition. In this case we speak about the flows coming out our first and last exclusive gateway. The second gateway acts as exclusive join which means that only the first flow that will arrive will cross. If instead we had a parallel gateway all flows should arrive before the process moved on.

```
<sequenceFlow id="flow15" name="" sourceRef="exclusivegateway2" targetRef="usertask2">  
  <conditionExpression xsi:type="tFormalExpression"><![CDATA[ ${redo==true} ]]>  
  </conditionExpression>  
</sequenceFlow>  
<sequenceFlow id="flow18" name="" sourceRef="usertask2" targetRef="exclusivegateway1">  
</sequenceFlow>  
<sequenceFlow id="flow19" name="" sourceRef="exclusivegateway1" targetRef="servicetask1">  
  <conditionExpression xsi:type="tFormalExpression"><![CDATA[ ${toUnit=="far"} ]]>  
  </conditionExpression>  
</sequenceFlow>  
<sequenceFlow id="flow20" name="" sourceRef="exclusivegateway1" targetRef="servicetask2">  
  <conditionExpression xsi:type="tFormalExpression"><![CDATA[ ${toUnit=="cel"} ]]>  
  </conditionExpression>  
</sequenceFlow>  
<sequenceFlow id="flow21" name="" sourceRef="exclusivegateway2" targetRef="endevent1">  
  <conditionExpression xsi:type="tFormalExpression"><![CDATA[ ${redo==false} ]]>  
  </conditionExpression>  
</sequenceFlow>
```

#### Code Snippet 19 : sequence flow refinement

Above we can see that just like in the forms the conditions are java code. In this case the clause we use must return a Boolean value.

#### 4.2.5 Deploying our Process

After we have concluded the designing and coding part of our process we must have ended up with a series of files. The only necessary file is a \*.bpmn20.xml that is our process model in bpmn2 format. During the deployment it's the only type of file that is evaluated and is the only factor for a successful deployment. Any other error will appear during execution. Of course if the model declares to use certain form files we must include those as during execution when requested to this files won't be found. In addition we must make sure that any class called by the process must exist in the class path of the engines web app. Finally the image file of the model is totally optional.

In our case we ended up with the following files:



- unit\_converter.bpmn20.xml
- converterForm.form
- redoForm.form
- unit\_converter.png

All we have to do now is to create a zip file name \*.bar for example converter.bar. Afterward we have to login with an administrator account to our manager app and go to the deployments page. Then we select upload and select the \*.bar file we want to deploy.

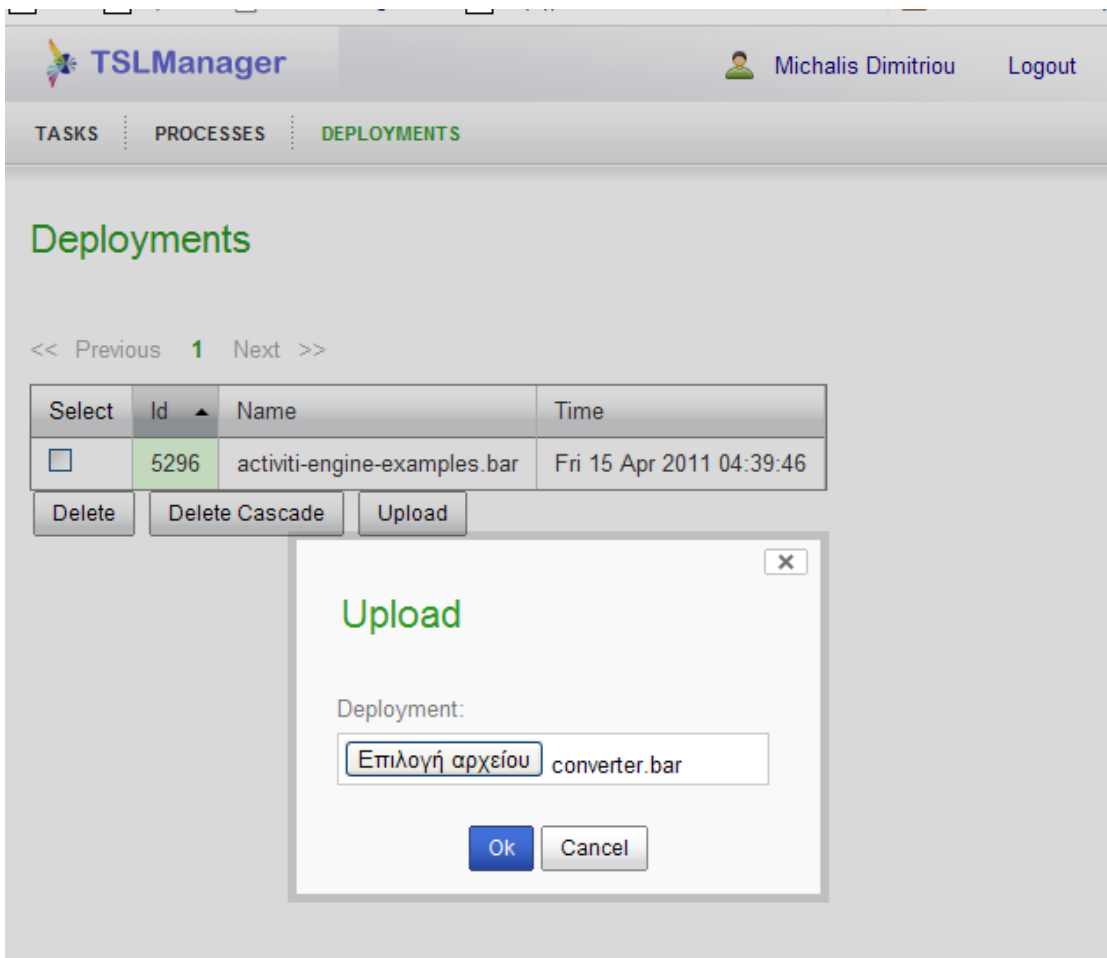


Figure 26 : uploading deployment

If all went well the new Deployment must appear in the Deployments and a new process in our processes tab.



## 4.2.6 Running the process

To Start the new process first we have to select “Start Process” for the process we are interested ( TempConverter ) in the Processes page. Then if we are logged in as “mdimitr” the first task should appear in “MyTask” at the Tasks page and selecting complete form should show the form for completion.

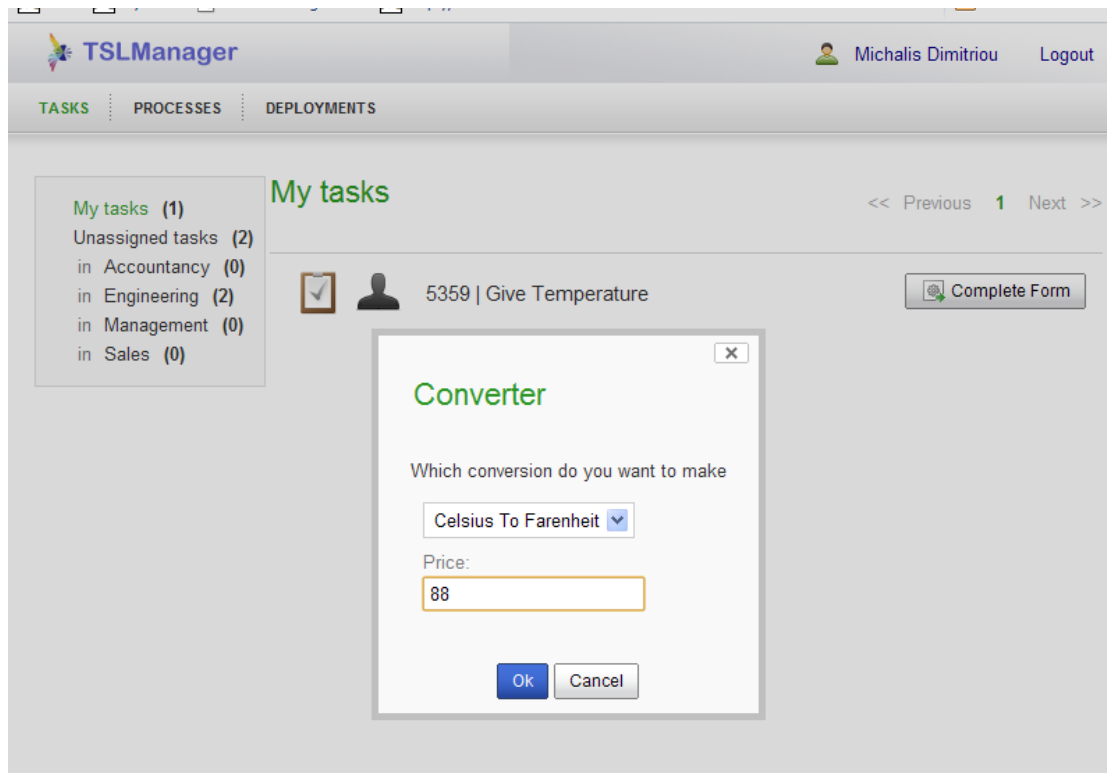


Figure 27 : Convert temperature form

We select to convert from 88 degrees Celsius to Fahrenheit. Then the engine should move to the service task to call the adequate operation of the web service. And if all worked a new Task should appear to see the result and choose to finish or convert again.



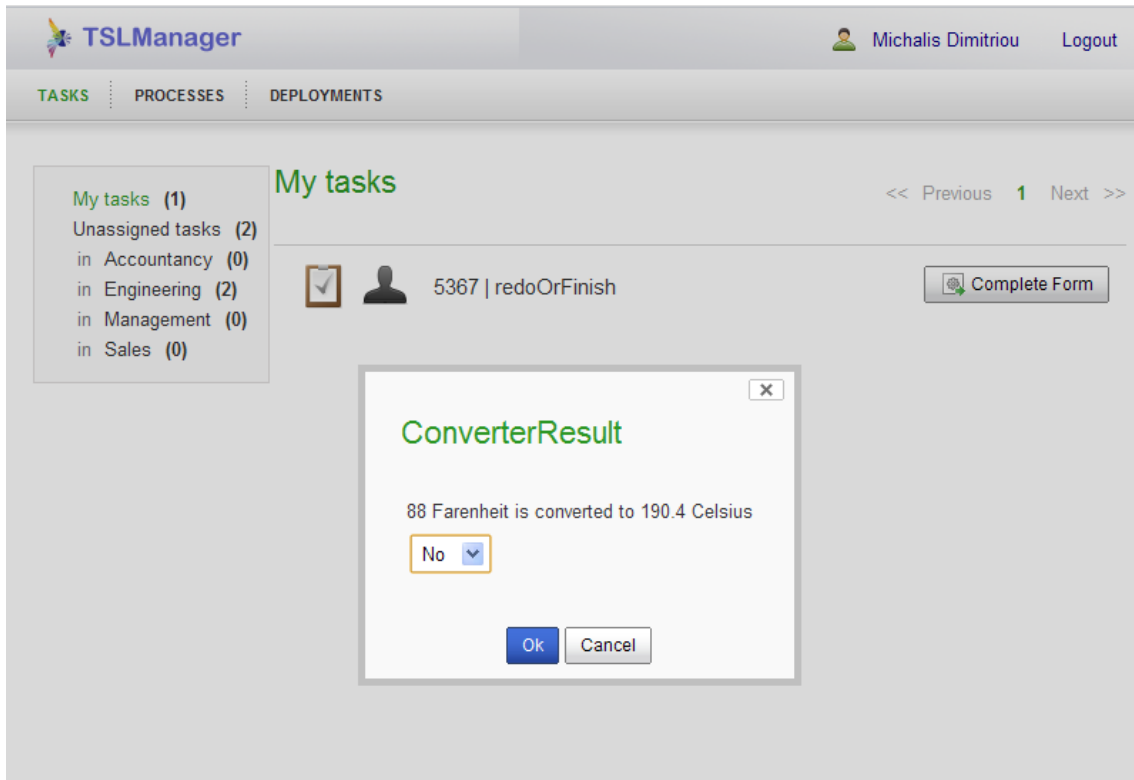


Figure 28 : Convert result form

If we choose “no” the process will finish otherwise the first task will appear again.

## 5 EXTENDING THE INFRASTRUCTURE TO RUN CHOREOGRAPHIES.

In this section we will try to describe our concept and way of thought so as to develop and evolve extensions for the activiti engine that will allow it to implement Choreographies. For the concept to be more easy understand will be described through a simple case study and try to fulfill the requirements needed step by step.

Although our Infrastructure supports by default orchestrations Choreographies is a totally different story. First of all we need to define the requirement of o Choreography to be considered in fact Choreography[16].



1. **Participants:** As we now Choreography is the coordinated interaction of two or more Orchestrations. So by definition we must have two or more separate Processes running.
2. **Distributed:** Our processes should be able to interact no matter where they are deployed. This means that they should work whether they exist in the same Infrastructure or not.
3. **Interoperability:** The processes should be able to interact no matter in what type of infrastructure they are deployed. To ensure this their communication must be consisted by open standards like SOAP through webservices or http request responses.
4. **No Centralized Control:** A proper choreography should not depend on a centralized control. All processes must communicate directly with each other.
5. **Privacy:** While to properly design our processes and have them successfully coordinate we have to have all of them in mind, each process should require only minimum information for its partners meaning that no detailed inside knowledge of a partner process should be required.
6. **Instantiation:** Every process must be able to have instances with separate scopes and access. This means that a process partner is an instantiation of a process type so that when its partner wants to contact it they contact the specific Instance on not any of that type. This was a major restriction we had so far with BPEL as we couldn't repetitively contact the same execution.

With the above in mind we must define and create the tools a process can have access to, in order for it to communicate and interact with other processes and thus form an choreography.

1. **Receive message:** A process must be able to wait for certain messages and once it has received them to continue its execution flow.
2. **Send messages:** A process must be capable to send a message to any of each partner.
3. **Instantiation:** We need a way to identify our partners. This means that we need Knowledge for their location and an identification that will separate them from the other instances of the same process type.



## 5.1 Existing Tools and workaround

### 5.1.1 Instantiation

As we have described before in our Infrastructure once a process is started a new process instance and a new execution are created. Those instances are given unique ids that we can use to access them. Here is some code that demonstrates how this can be done.

```
ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine()  
RuntimeService runtimeService = processEngine.getRuntimeService();  
Execution newExec = runtimeService.createExecutionQuery().processInstanceId(partnerId).singleResult();
```

#### Code Snippet 20 : get Process instance execution by process id

With the `ProcessEngines.getDefaultProcessEngine()` we get the main Process engine which as described is our root object. Through the `processEngine` object we can gain access to the `runtimeService` that can query through all the running process instances. As a result knowing our partner's id can give us access to its execution and the ability to interact with it.

### 5.1.2 Send messages

As is our Infrastructure doesn't support any way of actually sending a message to another process instance. To work around this setback we are going to use the **Java Service Task** artifact. As we described before the Java Service task can work for us as our Swiss army knife. We can practically use it to do anything by simply creating an adequate Java class and assign it to the task. With this in mind we can use it to read files query data from a database call Web Service or make http requests.

### 5.1.3 Receive message

Just like in the case of sending a message there is no artifact dedicated to such a function. What we can use to work around is the **receive task**. Although all it does is pause the execution, we can use it to receive information. What we need is for the process to wait until we have somehow fed it with the required information programmatically and then tell it to resume. This is done by sending a signal to the paused execution. Here is a sample of the code capable for that.





```
ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();  
RuntimeService runtimeService = processEngine.getRuntimeService();  
Execution newExec = runtimeService.createExecutionQuery().processInstanceId(PartnerId).singleResult();  
runtimeService.signal(newExec.getId());
```

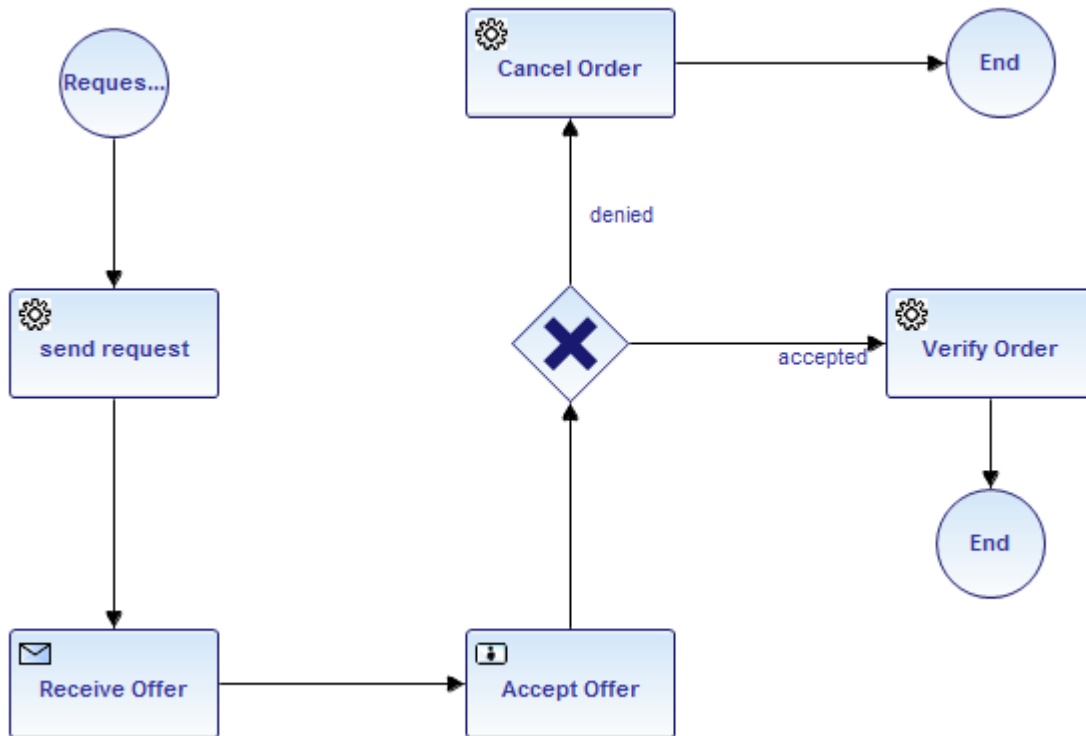
**Code Snippet 21 : send signal to execution**

As before, we access the execution of a process by its id and send a signal that will unpause it to the specific execution.

## 5.2 Case Study

Two evolve our thought on how we used a simple Choreography case study showing the interaction between a buyer and a seller during an order procedure. Here is a graphical representation of the two interacting processes.

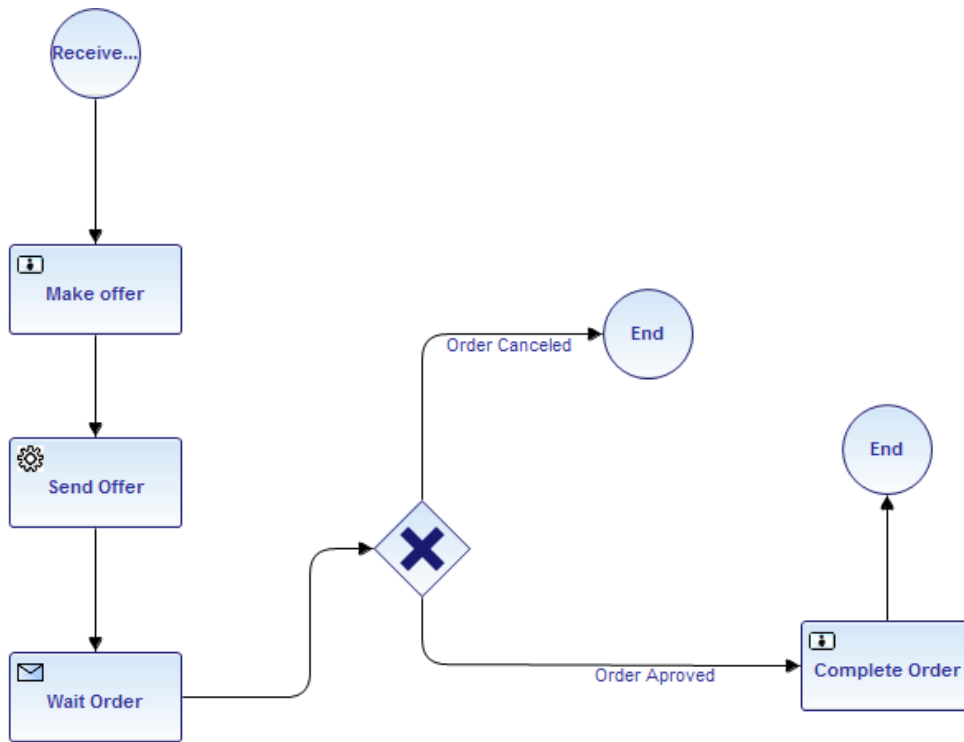




**Figure 29 : Buyer order process**

Above we can see the process that a buyer will follow. First he will request an offer over a certain item. The item is defined by a form assigned to the start event. After that he waits for the seller to send him an offer. Once he has received the offer he can assess it. If he likes it he can send that he accepts the offer otherwise he asks for a cancelation.





**Figure 30 : seller process**

This process gets instantiated after a call from a buyer. Once instantiated the seller is asked to make an offer and sends it. After that he waits for verification or cancelation if he receives cancelation the process ends. If he receives verification he is requested to continue with the order.

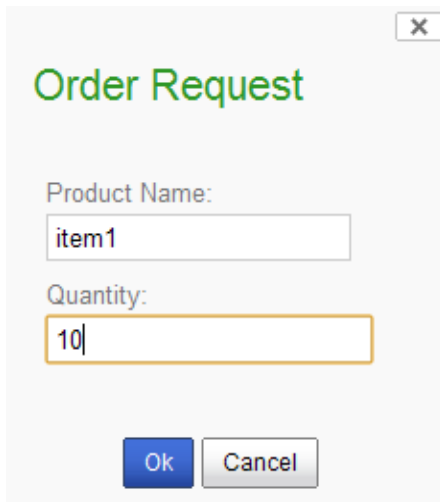
In both this processes we can see that in the place of a possible send message artifact we use the java service task. And for waiting for message the receive task respectively.

For the above processes to be executed we also need appropriate form files to interact with them. Here are the ones we developed for this case study.



```
<h1>Order Request</h1>
<table>
  <tr>
    <td>
      <label>
        Product Name:<br/>
        <input type="text" name="productName" value="" />
        <input type="hidden" name="productName_required" value="true" />
      </label><br/>
    </td>
  </tr>
  <tr>
    <td>
      <label>
        Quantity:<br/>
        <input type="number" name="quantity" value="1" min="1" />
        <input type="hidden" name="quantity_type" value="Integer" />
      </label>
    </td>
  </tr>
</table>
```

**Code Snippet 22 : Order Request form**



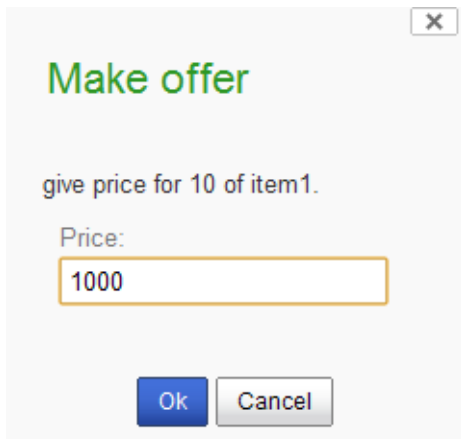
**Figure 31 : Order Request form**

In this form the buyer gives the name and the quantity of the product. Two variables are created one with the quantity of the product and the other with the products name.



```
<h1>Make offer</h1>
<p>
  give price for ${quantity} of ${productName}.
</p>
<table>
  <tr>
    <td>
      <label>
        Price:<br/>
        <input type="number" name="price" value="0" />
        <input type="hidden" name="price_type" value="String" />
      </label>
    </td>
  </tr>
</table>
```

**Code Snippet 23 : make offer form**



**Figure 32 : make offer form**

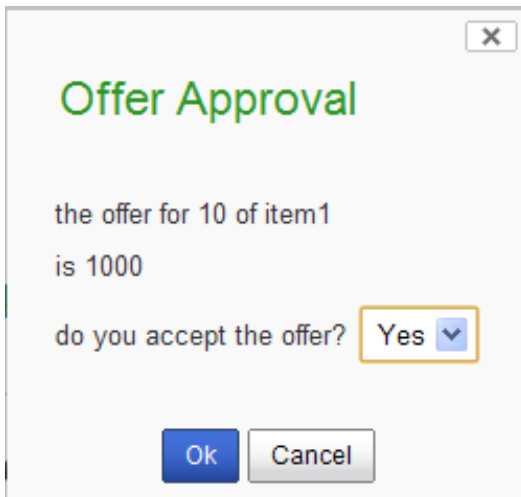
In this the seller makes an offer for the specified amount of a certain item. A variable “price” containing the amount is created.



```
<h1>Offer Approval</h1>
<p> the offer for ${quantity} of ${productName}</p>
<p> is ${price}
<p>
do you accept the offer?

<select name="acceptOffer">
  <option value="1">Yes</option>
  <option value="0">No</option>
</select>
<input type="hidden" name="acceptOffer_type" value="Integer" />
</p>
```

**Code Snippet 24 : assess offer form**



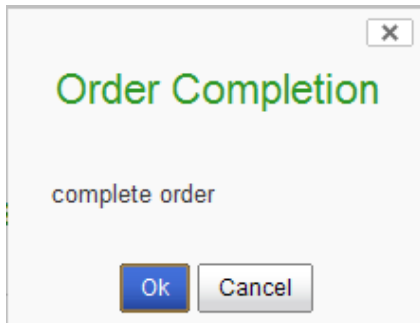
**Figure 33 : assess offer form**

Here the buyer assesses the offer and either accepts it or not.  
A Boolean variable “acceptOffer” is created.



```
<h1>Order Completion</h1>  
<p>  
  complete order  
</p>
```

**Code Snippet 25 : Complete order form**



**Figure 34 : Complete order form**

The seller receives the acceptance of the offer and completes it.



### 5.3 First Implementation same Engine multiple classes.

For our first attempt to implement our concept we are going to follow the simplest approach. First of all our two processes are deployed on the same Engine. This means that any java class we invoke can gain access to both processes and interact with them. Following this concept we will create a different class for every Service task in our Choreography. As a result we will end up with four delegate classes.

```
public class StartShopperProcess implements JavaDelegate {  
  
    public void execute(DelegateExecution execution) {  
        ProcessEngine processEngine =  
ProcessEngines.getDefaultProcessEngine();  
        RuntimeService runtimeService =  
processEngine.getRuntimeService();  
        IdentityService identityService =  
processEngine.getIdentityService();  
        ProcessInstance pi=null;  
        try {  
            identityService.setAuthenticatedUserId("mdimitr");  
  
            } finally {  
                identityService.setAuthenticatedUserId(null);  
            }  
        pi=runtimeService.startProcessInstanceByKey("shoperProcess");  
        Execution newExec =  
runtimeService.createExecutionQuery().processInstanceId(pi.getId()).singleResult();  
  
        String  
productName=(String)execution.getVariable("productName");  
        int  
quantity=((Integer)execution.getVariable("quantity")).intValue();  
        runtimeService.setVariable(newExec.getId(),"buyerId",  
execution.getProcessInstanceId());  
        runtimeService.setVariable(newExec.getId(),"productName",  
productName);  
        runtimeService.setVariable(newExec.getId(),"quantity",  
quantity);  
  
        execution.setVariable("shopperId", pi.getId());  
    }  
}
```

**Code Snippet 26 : Delegate class for sending a new order**

After the buyer has filled the order form the service task calls this class. Here we gain access, as described before, to the process engine and through it to the runtimeService. Through the runtime service we can instantiate a new seller process and get its process Id.





We store the Id in a variable of the local execution “shopperId”. Using this id we can query and access the execution off the newly instantiated seller process. This way we can create a variable there with the buyer’s process id. Furthermore we create two variables to store the quantity and the product name for our order.

As a result both processes know the id of their partner and we passed the data we wanted

After that the seller process will prompt him to fill the make offer form.

```
public class sendOffer implements JavaDelegate {
    public void execute(DelegateExecution execution) {
        ProcessEngine processEngine =
ProcessEngines.getDefaultProcessEngine();
        RuntimeService runtimeService =
processEngine.getRuntimeService();
        String shopperId=(String)execution.getVariable("buyerId");
        Execution newExec =
runtimeService.createExecutionQuery().processInstanceId(shopperId).singleResult();
        runtimeService.signal(newExec.getId());
        runtimeService.setVariable(newExec.getId(), "price", execution.getVariable("price"));
    }
}
```

**Code Snippet 27 : Delegate class for sending the offer**

After the make offer form is filled the following service task calls this class. Here we use the store id of the buyer’s process instance and use it to access it and create a new variable to it containing the amount of the offer. Furthermore we send a signal to the buyers execution to move on from the receive task that it has paused.

After that the buyer is prompt to fulfill the asses offer form. Depending on the choice a different class will be called and as in send offer class we pass the choice made through a variable.

Although this implementation is fully functional and seems to fulfill most of our demands for choreography it has a major flaw. It runs on a single Engine and as mentioned our demand for Choreography is distribution which means they must be able to run in different machines. With this implementation this is not feasible. Yet with few changes we can evolve it to work between two engines.



## 5.4 Second Implementation different Engine multiple services.

What we need to do is to instead of using a class to interact with both partners to break this into a class and a web service or a servlet. For easier understanding we are going to use http requests to communicate.

With this concept in mind for every interaction between two partners we require a delegate class that will make the request and a servlet or a web service that will receive it. We assume that we know the location of the partner Process and have incorporated in our model.

Re out that all the calling classes will be almost identical and that will change will be the http parameters passes and the result variable.

This means that just as we acted with our orchestrations and created a single delegate class to make all our Web Service calls similarly we can make one for our http call.

This class will look like that:

```
public class WsRestDelegate implements JavaDelegate {

    private Expression Url;
    private Expression parameters;
    private Expression returnValue;
    public void execute(DelegateExecution execution) {

        String URLString = (String)Url.getValue(execution);
        String paramStrings = "";
        if (parameters!=null) {
            StringTokenizer st = new StringTokenizer(
(String)parameters.getValue(execution), ",");
            while (st.hasMoreTokens()) {
                paramStrings=paramStrings+"&"+st.nextToken().trim();
            }
        }
        String result = null;
        if (URLString.startsWith("http://"))
```



```
    {
        try
        {
            String urlStr = URLString;
            if (paramStrings.length () > 0)
            {
                urlStr += "?" + paramStrings;
            }
            URL url = new URL(urlStr);
            URLConnection conn = url.openConnection ();
            BufferedReader rd = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
            StringBuffer sb = new StringBuffer();
            String line;
            while ((line = rd.readLine()) != null)
            {
                sb.append(line);
            }
            rd.close();
            result = sb.toString();
        } catch (Exception e)
        {
            e.printStackTrace();
        }
        if (returnValue!=null) {
            String returnVariableName = (String)
returnValue.getValue(execution);
            execution.setVariable(returnVariableName, result);
            System.out.println(" returned "+returnVariableName+" = "+
result);
        }
    }
}
```

**Code Snippet 28 : delegate class to make Http calls**

The above class works exactly like our webService call class worked instead now we make an Http request.

To level with the previous implementation this class will call a service that will instantiate the sellers process, store the http parameters as local variables and send with its response its process id.

On our next calls we will sent as parameter the process id for the receiving service to access the right process instance.

On the receiving point the service will look like That:



```
protected void executeWebScript(ActivitiRequest req, Status status,
Cache cache, Map<String, Object> model) {

    ProcessEngine processEngine =
ProcessEngines.getDefaultProcessEngine();
    RuntimeService runtimeService = processEngine.getRuntimeService();
    IdentityService identityService =
processEngine.getIdentityService();
    ProcessInstance pi=null;
    try {
        identityService.setAuthenticatedUserId("mdimitr");

    } finally {
        identityService.setAuthenticatedUserId(null);
    }

    pi=runtimeService.startProcessInstanceByKey("shopperProcess");

    Execution newExec =
runtimeService.createExecutionQuery().processInstanceId(pi.getId()).singleResult();
    String productName=(String)req.getString("productName");
    int quantity=Integer.parseInt(req.getString("quantity"));
    int callerId=Integer.parseInt(req.getString("buyerId"));
    runtimeService.setVariable(newExec.getId(),"buyerId",
callerId);
    runtimeService.setVariable(newExec.getId(),"productName",
productName);
    runtimeService.setVariable(newExec.getId(),"quantity",
quantity);
    model.put("shopperId", pi.getId());
}
```

### Code Snippet 29 : Start Shopper webScript

We can see that it is very similar to our new order class in snippet 26. The basic Difference is that the variables from the caller are now http parameters and in order to create a new variable to the caller we must include it in our response.

Similarly we form receiving WebScript for every receive task.

## 5.5 Final Implementation Different Engine, single receiving services.

When creating such infrastructures we must always have in mind that they are going to be used by mostly non developers. As a result although our last implementation confronts



fully with our choreography requirements is user unfriendly. Every time we create a new process we must create adequate receiving WebScript and this is quite redundant. If we observe all our web script we can see huge similarities and a pattern for a single global receiver emerges.

This receiver will have the following logic. It will check for a parameter with name “ProcKey” this will be the key of a process that must be instantiated. So if it exist it will instantiate the mentioned process.

Otherwise it will check for the parameter with name “procId”. This will be the id of the process instance called.

After that it will check for a parameter with name “return”. This parameter will hold the names of variables required to be returned, delimited with a minus symbol “-”. Finally it will browse through all the rest parameters and instantiate them as Variables.

This global receiver will look like that:

```
protected void executeWebScript(ActivitiRequest req, Status status,
Cache cache, Map<String, Object> model) {
    HttpServletRequest request=req.getHttpServletRequest();
    Enumeration<String> names = request.getParameterNames();
    String name=" ";
    Execution newExec=null;
    ProcessEngine processEngine =
ProcessEngines.getDefaultProcessEngine();
    RuntimeService runtimeService = processEngine.getRuntimeService();
    ProcessInstance pi=null;
    String returnVars;
    String procId;
    String procKey=req.getString("procKey");
    if (procKey!null){

        pi=runtimeService.startProcessInstanceByKey("shoperProcess");
        newExec =
runtimeService.createExecutionQuery().processInstanceId(pi.getId()).singleResult();
    }
    else{
        procId=req.getString("procId");
        newExec =
runtimeService.createExecutionQuery().processInstanceId(procId).singleResult();
    }

    while((name=names.nextElement())!null){
        if(name!="procKey" && name!="procId" && name!="return"){
            runtimeService.setVariable(newExec.getId(),name,
req.getString(name));
        }
    }
}
```



```
returnVars=req.getString("return");  
String[] variables=null;  
  
if(returnVars!=null){  
    variables=returnVars.split("-");  
}  
  
for(int i=0; i < variables.length; i++){  
    model.put(variables[i],  
runtimeService.getVariable(newExec.getId(),variables[i]));  
}  
}
```

**Equation 30 : global receiver WebScript**

## 5.6 Extension Summary

What we developed are classes and native services that provide the ability for independent processes to communicate and interact with each other thus forming choreographies. Furthermore through our case study we demonstrated a way of thinking conceptualizing and designing orchestrations to interact with each other so as to lead to functional and useful choreographies. Such functionality can be incorporated in the activity engine and other similar tools and be perfected through real life applications. What is required is standardization of communication protocols between processes and lift the mist of what choreographies really are as many interpret and view it in different ways.



## **CONCLUSIONS AND FUTURE WORK**

During the study for this work we gain significant knowledge on how to develop and run web services and WebService Orchestration. Additionally we noticed the limits of current frameworks and infrastructures. This fact pushed the community to the creation of BPMN 2 a business model that can satisfy both developers and managers. Believing that choreographies are an integral part of the future of business processes we attempted to figure out if the latest developments can provide easy implementation. As it was figured out Choreographies seem to most developers still very exotic and hesitate to incorporate them in their work. Yet it is Obvious that we are moving away from strictly orchestration character that was set by BPEL. As a result the artifacts for a choreography capable infrastructure exist. The result of this study was an infrastructure capable of providing an environment to easily develop both orchestrations and choreographies.

In the future we must test the logic and the stability of this implementation with more complex models. Furthermore we can add more BPMN 2 artifacts that will give significant more possibilities to the model designers. A first improvement that was notices is to store the calls/messages so as if a call is made before a receive task is reached to pass it without pausing. In addition to this, calls can also have receive task targets so as if a receive task is never reached the call to be discarded.



## REFERENCES

5. ["Web Service to Web Service Communication"](#). Retrieved 2011-09-22.
6. ["Web Services Description Language \(WSDL\) Version 2.0 Part 1: Core Language"](#). Retrieved 2011-09-22.
7. ["Web Services Glossary"](#) . W3C. February 11, 2004. Retrieved 2011-09-22.
8. ["XML 1.0 Specification"](#). W3.org. Retrieved 2011-09-22.
9. Allweyer T (2010 Feb 22). [BPMN 2.0 - Introduction to the Standard for Business Process Modeling](#). BoD. ISBN 978-3-8391-4985-0.
10. [apache tomcat server](#)
11. BPEL vs BPMN 2.0: Should you care? Frank Leymann [http://bpt.hpi.uni-potsdam.de/pub/BPMN2010/Program/bpmn2010\\_leymann.pdf](http://bpt.hpi.uni-potsdam.de/pub/BPMN2010/Program/bpmn2010_leymann.pdf)
12. Briol P. (2008 April 12). *BPMN, the Business Process Modeling Notation Pocket Handbook*. LuLu. ISBN 978-1-4092-0299-8.
13. Briol P. (2010 Nov 16). *BPMN 2.0 Distilled*. LuLu. ISBN 978-1-4461-0406-4.
14. [Business Process Execution Language for Web Services, Version 1.1 \(PDF\)](#) (5 May 2003)
15. *Business Process Modeling Notation*, specification of BPMN v1.0 by Stephen A. White (3 May 2004), for Business Process Management Initiative (BPMI)
16. Chris Peltz: [Web Services Orchestration and Choreography](#). IEEE Computer (COMPUTER) 36(10):46-52 (2003)
17. Chun Ouyang, Marlon Dumas, Arthur H. M. Ter Hofstede : From Business Process Models to Process-oriented Software Systems: The BPMN to BPEL Way
18. Debevoise, Neilson T, et al. (2008 July 4). *The MicroGuide to Process Modeling in BPMN*. BookSurge Publishing. ISBN 978-1-4196-9310-6.
19. Fielding, Roy T.; Gettys, James; Mogul, Jeffrey C.; Nielsen, Henrik Frystyk; Masinter, Larry; Leach, Paul J.; Berners-Lee (June 1999). ["RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1"](#).
20. FreeMarker template engine <http://freemarker.sourceforge.net/>
21. Gero Decker, Oliver Kopp, Frank Leymann, Mathias Weske: [BPEL4Chor: Extending BPEL for Modeling Choreographies](#). ICWS 2007:296-303





22. Grosskopf, Decker and Weske. (2009 Feb 28). [\*The Process: Business Process Modeling using BPMN\*](#). Meghan Kiffer Press. ISBN 978-0929652269.
23. Jack Vaughan: [BPMN 2.0 adds notation to handle BPM choreography](#). SearchSOA.com
24. Johannes Maria Zaha, Alistair P. Barros, Marlon Dumas, Arthur H. M. ter Hofstede: [Let's Dance: A Language for Service Behavior Modeling](#). OTM Conferences 2006:145-162
25. Object Management Group (OMG) <http://www.omg.org/>
26. Pautasso, Cesare; Zimmermann, Olaf; Leymann, Frank (2008-04), "[RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision](#)", *17th International World Wide Web Conference (WWW2008)* (Beijing, China)
27. [Process Modeling Notations and Workflow Patterns](#), paper by Stephen A. White of IBM Corporation (2006)
28. Richardson, Leonard; Ruby, Sam (2007-05), [RESTful Web Services](#), O'Reilly, ISBN 978-0-596-52926-0
29. Ryan K. L. Ko, Stephen S. G. Lee, Eng Wah Lee (2009) Business Process Management (BPM) Standards: A Survey. In: Business Process Management Journal, Emerald Group Publishing Limited. Volume 15 Issue 5. ISSN 1463-7154. [PDF](#)
30. S-Cube Knowledge Model: [Service Choreography](#)
31. The Activiti project <http://activiti.org>
32. The ms-bpel standard by OASIS [oasis ms-bpel](#).
33. the Spring Surf Project <http://www.springsurf.org/>
34. The Organization for the Advancement of Structured Information Standards (OASIS) [Oasis-open.org](#).
35. W3C (September 2011). "[World Wide Web Consortium \(W3C\) About the Consortium](#)". Retrieved 2011-09-22.
36. [Web Services Choreography Working Group](#) at W3



37. White, Stephen A, and Miers, Derek (2008 August 28). *BPMN Modeling and Reference Guide*. Future Strategies Inc.. [ISBN 978-0-9777-5272-0](#).
38. Wikipedia contributors. Unified Modeling Language. Wikipedia, The Free Encyclopedia. November 4, 2011, 15:44 UTC. Available at: [http://en.wikipedia.org/w/index.php?title=Unified\\_Modeling\\_Language&oldid=458986043](http://en.wikipedia.org/w/index.php?title=Unified_Modeling_Language&oldid=458986043). Retrieved 2011-09-22
39. Zongyan Qiu, Xiangpeng Zhao, Chao Cai, Hongli Yang: [Towards the theoretical foundation of choreography](#). WWW 2007:973-982

