NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ANALYSIS OF DATA-DRIVEN WEB APPLICATION VERSUS PROCESS-DRIVEN APPLICATION

by

Turki Abdullah A. Almutairi

December 2018

Thesis Advisor:	Glenn R. Cook
Second Reader:	Arijit Das

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** December 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE ANALYSIS OF DATA-DRIVEN WEB APPLICATION VERSUS PROCESS-DRIVEN APPLICATION	**5. FUNDING NUMBERS**
6. AUTHOR(S) Turki Abdullah A. Almutairi	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Enterprise applications are a type of software used in organizations to provide the functions for one or more business domains. The standard approach to developing enterprise application is using common programming platforms like Java EE or Microsoft .NET. However, in recent years new enterprise applications development platforms called Business Process Management Systems (BPMSs) have emerged. BPMS providers claim that their technologies allow organizations to develop enterprise applications faster than by using traditional applications development platforms. No comprehensive comparison of using the two approaches has yet been conducted. The purpose of this study is to compare two enterprise application features, persistence and messaging, available in both technologies—here, Java EE and Bonita BPM. The findings of this study revealed that developing applications using Bonita BPM takes less effort than using Java EE. However, facilitating applications development with Bonita BPM comes with a cost, which is that it limits the developer's ability to use specific preconfigured persistence and messaging technology.

14. SUBJECT TERMS business process management, process centric organizations	**15. NUMBER OF PAGES** 87
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

THIS PAGE INTENTIONALLY LEFT BLANK

ANALYSIS OF DATA-DRIVEN WEB APPLICATION VERSUS PROCESS-DRIVEN APPLICATION

Turki Abdullah A. Almutairi
Major, Army, Saudi Arabia
Bachelor of Science, Taif University, 2007

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
December 2018

Approved by: Glenn R. Cook
Advisor

Arijit Das
Second Reader

Dan C. Boger
Chair, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Enterprise applications are a type of software used in organizations to provide the functions for one or more business domains. The standard approach to developing enterprise application is using common programming platforms like Java EE or Microsoft .NET. However, in recent years new enterprise applications development platforms called Business Process Management Systems (BPMS) have emerged. BPMS providers claim that their technologies allow organizations to develop enterprise applications faster than by using traditional applications development platforms. No comprehensive comparison of using the two approaches has yet been conducted. The purpose of this study is to compare two enterprise application features, persistence and messaging, available in both technologies—here, Java EE and Bonita BPM. The findings of this study revealed that developing applications using Bonita BPM takes less effort than using Java EE. However, facilitating applications development with Bonita BPM comes with a cost, which is that it limits the developer's ability to use specific preconfigured persistence and messaging technology.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

API	Application Programming Interface
BAM	Business Activity Monitoring
BDM	Business Data Model
BPM	Business Process Management
BPMN	Business Process Management Notation
BPMS	Business Process Management systems
CMS	Content Management System
CRM	Customer Relationship Management
EJB	Enterprise Java Beans
IBM	International Business Machine
IDE	Integrated Development Environment
Java EE	Java Enterprise Edition
JMS	Java Messaging Service
JPA	Java Persistence API
JSF	Java Server Faces
MSMQ	Microsoft Message Queuing
OOP	Object Oriented Programming
ORM	Object-Relational Mapping
POJO	Plain Old Java Object
SQL	Structured Query Language
XML	Extensible Markup Language

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

My parents, I am indebted to thank you for your love and support.

My wife, Najah, and my daughter, Reham, thank you for your patience, encouragement and love.

Dr. Dan Boger, the faculty staff and members, thank you for all your support and encouragement.

Mr. Glenn Cook, thank you for your wonderful support, your helpful advice and your pleasant supervision.

Mr. Arijit Das, thank you for your informative discussions, valuable hints and for your encouragement.

My wonderful colleagues in Ministry of Interior of Saudi Arabia, thank you all for your support.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM OVERVIEW

The rapidly evolving nature of the business world requires that companies and organizations be adaptable to changes, including fluid customer requirements, stiff market competition, and changing regulations. Changes in the business environment affect the applications and software that enterprises use to perform their business because these applications must meet their requirements for change. One particularly important type of software is an enterprise application, a type of software that is used across the functional departments of an organization to support the performance of its business. Such applications can be obtained by purchasing software packages that are ready-made and standard or by building customized software solutions. Some organizations may prefer to build enterprise applications that are tailored to their specific requirements and support their growth.

The creation of customized enterprise applications takes time and requires a team of specialized programmers and analysts, as well as an enterprise software development platform. There are many applications development platforms, each providing businesses with different capabilities to create, test, and run enterprise applications. The Microsoft .NET platform and Java Enterprise Edition (Java EE) platform are two examples out of many recently available development platforms.

Alternatively, a newer technology that can also be used to build enterprise applications is Business Process Management Systems (BPMSs). Currently, there are many BPMS products available commercially, such as IBM Business Process Manager and Oracle Business Management Suite, as well as open-source versions, including Bonita BPM and JBPM. Applications built with BPMSs are called process-centric applications or business process applications (Allègre, 2013). In brief, in order to design process-centric applications, BPMSs provide a development environment that allows developers to visually construct the flow of the business process in terms of sequenced activities; BPMSs then enable the developers to adjust custom settings and script codes for each activity. BPMSs also allow connectivity with external applications and data sources.

Thus, both technologies—BPMS and a traditional enterprise applications development platform like Java EE—can be used to build enterprise applications. Currently, though, BPMS has been less widely adopted by businesses than traditional platforms despite BPMS vendors' claims that their products allow developers to build enterprise applications faster than they could using a traditional application development platform like Java EE. A study by Capgemini (2014) indicates that Pega BPM reduces the amount of development time required to build an application when compared to using Java EE. Another BPMS vendor called Appian announces that their Appian BPM product helps developers to rapidly build enterprise applications (Appian, n.d.). However, no rigorous third-party study of the differences between these methods—traditional development platforms and BPMS—has yet been performed, which leaves open a significant question: namely, what are the most important differences between building an application by using BPMS vs. more common development platforms? Capgemini's and Appian's statements also lead to another question: why, given their supposed capabilities, do BPMSs not dominate the applications development landscape?

These questions motivate the work in this study. To answer these questions, this study focuses on analyzing the differences between an enterprise application built using Java EE and a process-centric application built with a BPMS, Bonita BPM. Both applications perform two common functions of enterprise applications, persistence and messaging, selected because they are essential components of enterprise applications.

B. RESEARCH OBJECTIVES

The purpose of this qualitative study is to offer a case study to compare the functionality that BPMS provides compared to a traditional programming platform. Bonita BPM is used to develop a process-centric application that performs persistence and messaging functions, while Java EE is used to develop a traditional enterprise application offering comparable functionality. This study then analyzes these applications to determine whether the BPMS's capabilities reduce the amount of effort needed to develop applications , which would in turn reduce the amount of effort needed to adapt them to increasing functional requirements. The study focuses on the technical aspects of BPMS compared to traditional application development techniques—a comparison that contributes to previous studies' findings on this subject from a technical standpoint.

C. RESEARCH QUESTIONS

How can Business Process Management Systems (BPMS) impact organizations' ability to adapt to continously changing functional requirements?

In order to answer this question, the thesis will address the following questions:

1. What are the key differences between a BPMS and a traditional custom application development platform?

2. Can using BPMS reduce the development effort required to adapt enterprise applications to functional requirements?

3. What are the limitations of BPMS as opposed to custom applications development frameworks?

D. LITERATURE REVIEW

BPMS is a technology that aims to allow organizations to automate their business process. It provides a set of tools that allow software developers and business analysts to design, build, test, and deploy business process applications. According to Smith and Fingar (2003) in *BPM: The Third Wave*, BPM is a management methodology that enables organizations to create and improve business processes. Similarly, Grafzig, Banke, and Salma (2005), in their book *Enterprise SOA Service-Oriented Architecture Best Practices*, indicate that Business Process Management Systems (BPMS) is a technology that implements BPM methodology concepts and enables the creation, modeling, implementation, and monitoring of business processes.

Recently, many BPMS vendors have emerged, including Bonita BPM, a BPMS made by Bonitasoft. Bonita BPM is used to develop business process applications that can be implemented and used on an enterprise scale. Developing an application using BPMS is different from using a traditional programming language because it uses different standards. Developing a business process application using BPMS involves representing business process flows graphically based on Business Process Management Notation (BPMN) standard rather than using traditional code (Allègre, 2013). On the other hand, developers can also use conventional programming languages to develop applications. A developer is required to use multiple programming and scripting languages to create and test different application components. To support developers, the application development platforms provide programmers with an integrated development environment (IDE) to create, design, and test applications. Java EE or Microsoft .NET are examples of conventional applications development platforms that enable developers to build scalable and mission-critical enterprise applications.

Showing differences between features that exist in both technologies is a way to find an answer the question of which technology is better to develop enterprise applications. Regarding this question, academic studies come to varying conclusions. A study conducted by a consultancy firm called Capgemini measured developers' productivity when using Pega 7 BPMS versus Java EE to build an enterprise application. Based on their study, Pega 7 BPMS' productivity factor, which measures the development effort, is 6.4 times higher than that of Java EE (Capgemini, 2014). A related study by Cui and Liu (2010) demonstrates the impact of BPMS on organizations, examining the factors that influence the adoption and diffusion of BPMS in organizations. In their study, based on user interviews, they found that the characteristics of integrating and streamlining business processes encourages BPMS adoption and diffusion. However, their study does not address in detail how BPMSs integrate technically into business processes. Likewise, Yeshanew and Mapinduzi (2010) address the impacts of BPMS on users in organizations, but they also do not investigate the technical aspects of that impact. In their recommendations for future work, they suggest exploring the impact of BPMS from a technical point view.

To take these studies further, therefore, the present study addresses the integration characteristics that BPMS has through its messaging feature. We also compare the BPMS messaging feature with Java EE's messaging feature to find the differences between them.

Likewise, we explore some of BPMS's technical aspects to explain how BPMS may reduce the effort needed to develop applications.

E. METHODOLOGY

The methodology used to answer the questions posed by this study is a comparison between two features available in both traditional development platforms—here, Java EE—and BPMS. The first feature is persistence, and the second feature is messaging. In this study, we compare a Java EE persistence application that uses Java Persistence API (JPA) with the persistence feature in a Bonita BPM-based application. Also, we compare a Java EE application that uses Java Messaging Service (JMS) with a Bonita BPM-based application that uses process messaging. Then, we analyze the differences between the messaging and persistence features in Java EE and Bonita BPM applications in terms of effort—the number of programming steps and amount of configuration needed by both technologies, which in turn affects the time needed for developers to build applications.

We used Java EE as a traditional applications development platform for the following reasons:

- Java is a high-level programming language used to build reliable and secure extended applications for enterprises (Oracle, n.d.).
- Java is built based on the principle of "develop once, run everywhere," which makes Java applications independent and able to run on different operating systems (Oracle, n.d.).
- Java provides a multi-tier enterprise application model, which supports flexibility in applications development (Oracle, n.d.).

We use BonitaSoft's Bonita BPM as a BPMS for the following reasons:

- Bonita BPM is an open-source BPMS that allows developers to build process-centric applications (Bonitasoft, 2009).
- Bonita BPM has uses all common BPMS standards, making it a representative BPMS.

- Bonita BPM has detailed documentation that serves the purpose of this research.

F. SUMMARY

In this chapter we have defined the current argument about whether BPMS reduces enterprise application development effort. Although BPMS vendors claim that BPMSs facilitate the development of enterprise applications, most developers still use traditional platforms for applications development, which raises the question of why BPMS systems are not very common as a main enterprise applications development platform, which in turn was the main motivation for this study. This question can be answered from a technical standpoint by investigating the differences between these two approaches.

The remainder of this thesis is divided into four chapters. Chapter II introduces the Bonita and Java EE technologies. Chapter III describes the process of building the persistence and messaging applications using both platforms. Chapter IV contains the analysis of the Java EE–based application versus the Bonita BPM–based application. Chapter V contains the comparison results and recommendations for future work.

II. JAVA EE AND BONITA BPM

This chapter offers background on the two platforms used to develop the applications compared in this study: Java EE and Bonita BPM. Each section includes a general introduction to the platform followed by a discussion of the relevant features—persistence and messaging—used to construct the applications.

A. JAVA EE PLATFORM

Java EE is a development environment for building and running wide, multi-tier, scalable, reliable, and secure network applications using Java programming language (Jendrock et al., 2013c). In particular, the Java EE platform provides Applications Programming Interfaces (APIs), a set of technologies that enable programmers to build high-efficiency, enterprise-class applications (Jendrock et al., 2013d). The following are some of the Java APIs:

- Enterprise Java Beans (EJB) Technology is a technology in the Java EE platform to enable developers to write business logic (Jendrock et al., 2013d).
- Java Servlets Technology is used to develop classes to handle to extend applications hosted on web servers (Jendrock et al., 2013d).
- Java Server Faces (JSF) Technology provides a framework for user interfaces in web applications (Jendrock et al., 2013d).
- Java Messaging Service (JMS) allows applications written in Java to create, receive, and read messages while interacting with other systems (Jendrock et al., 2013d).
- Java Persistence API (JPA) is a standard designed to enable applications written in Java to store data in databases by using object-relational mapping (ORM) (Jendrock et al., 2013d).

- Java Mail API provides the ability to send and receive e-mail messages via the Java application (Jendrock et al., 2013d).

1. Java Persistence API

This section provides general overview about the Java Persistence API used for persistence in Java EE.

a. Background

Data persistence is a concept of allowing data to be exist in a fixed state in a storage system and be available to be accessed in later time even beyond the end of the life span of the process which created it (Sun, 2016). Data persistence is a necessary for applications because applications create, read, write and update data. Databases is one of many technologies which is used as a storage system to keep data persisted (Sun, 2016).To allow applications to persist the data into databases it required a persistence technology. JPA is a Java EE technology used for data persistence. This section gives a brief view about JPA.

Java JPA API is a Java EE specification used to persist data by using a mechanism called Object Relational Mapping (ORM) (explained in chapter 3 section A) (Stancapiano, 2017). JPA does not perform object mapping by itself it is a specification which tells how to implement map objects into relational database (Stancapiano, 2017). ORM provider is a software which implements JPA specification. Hibernate is a JPA compatible open source software uses ORM mechanism.

Based on JPA specification ORM mechanism could be implemented by either done using annotations or setting Extensible Markup Language (XML) files inside the project . In this study we used XML method to map object's properties to a relational data table. We used Hibernate as ORM provider to persist data because it is a very common to developers.

b. Persistence requirements using XML method

The following steps are the general steps required to persist data in a relational database by using Hibernate. We used XML configurations approach to map the Java class properties to the relational data table columns.

(1) defining Plain Old Java Class

Plain Old Java Class (POJO) is a regular Java class which is required to be mapped into a relational data table. It has to contain at least one object constructor and private getters and setters.

(2) ORM Mapping provider

We used Hibernate ORM to map the POJO class properties to their proper column in the relational data table.

(3) Configure mapping XML file

Hibernate uses an XML file for mapping. It contains the POJO class properties names associated with the data table columns name

2. Java Messaging

This section provides general overview about the Java Messaging Service JMS API used for messaging in Java EE.

a. background

An enterprise application is a piece of software distributed to an organization's sections. Enterprise applications consist of several applications, including sales, finance, and supply management, that exchange information among each other: for example, a financial application for payroll management may include information about employees that is exchanged with a human resources application.

Information exchange between enterprise applications makes it easier to for organizations to do business. Allowing enterprise applications to communicate through messaging facilitates information exchange between enterprise applications, especially between enterprise applications that have different architectures. Take an enterprise application built using C++ programming language: such an application will differ in its architecture from other enterprise applications built using Java programming language. Such difference makes it difficult for those applications to exchange information.

Java Messaging Service (JMS) technology was developed to resolve this issue. Messaging is a mechanism to exchange information between disparate enterprise applications, and JMS is a Java EE technology that enables enterprise applications messaging. The integration of enterprise applications requires interoperability throughout a messaging system that manages the messaging process between different applications regardless of their architecture (Richards at el., 2009). In other words, a messaging service acts as an intermediary between the various enterprise applications to facilitate the messaging process between them. Enterprise applications do not communicate directly with the messaging service but use specific messaging interfaces for this purpose.

Of the various technologies offered by the Java EE platform, JMS allows applications to send and receive messages through interfaces and classes of software (Jendrock et al., 2013d). Figure 1 illustrates general view of how enterprise applications use JMS to communicate. In this process, application A uses the messaging application program interface API to generate a message and then sends it to a message-oriented middleware. The message-oriented middleware will then handle and deliver the message to application B. The same messaging process occurs in both directions—A to B and B to A. In this study we used Active MQ messaging provider in Java EE messaging application.

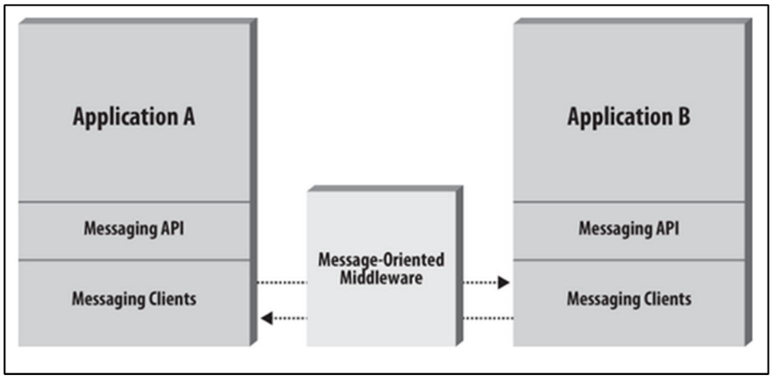


Figure 1. Message-oriented middleware. Source: Java Message Service (Richards et al., 2009).

b. ActiveMQ Messaging Middleware

Different JMS-compliant messaging middleware applications are available in the market today, either in a commercial or open-source format. Messaging middleware optimized for business-class categories includes International Business Machine (IBM) Websphere MQ and Microsoft Message Queuing (MSMQ), while those optimized for open-source messaging include ActiveMQ and RabbitMQ. The messaging middleware used in our study is ActiveMQ. We used the version of ActiveMQ that comes already installed in an application server called Wildfly. ActiveMQ is preconfigured and has been integrated into the Wildfly application server which make it easier to develop the Java EE messaging application.

c. Application Server

According to Footen and Faust (2008), "an application server is a specific type of software platform on which an enterprise can build and host applications for its users" (p. 154). In our application, we used the Wildfly application server to host and run the Java EE messaging application. We used Wildfly application server because it is compliant with Java EE specifications (Wildfly, 2017).

d. Asynchronous One-to-One Messaging

We developed a simple Java EE messaging application that uses JMS API. The application sends the message from one source to one destination. According to Richards et al. (2009), JMS technology provides two messaging models: one-to-one and publish/ subscribe. Richards and his colleagues explain that in the one-to-one messaging model, a client sends a message that is placed in a queue until the receiver retrieves the message, at which time messaging sends the first client an acknowledgment that the message has been read. In the one-to-one messaging model, there is only one sender, who sends a message to one receiver (Richards et al., 2009). Figure 2 shows a representation of one-to-one messaging in JMS.

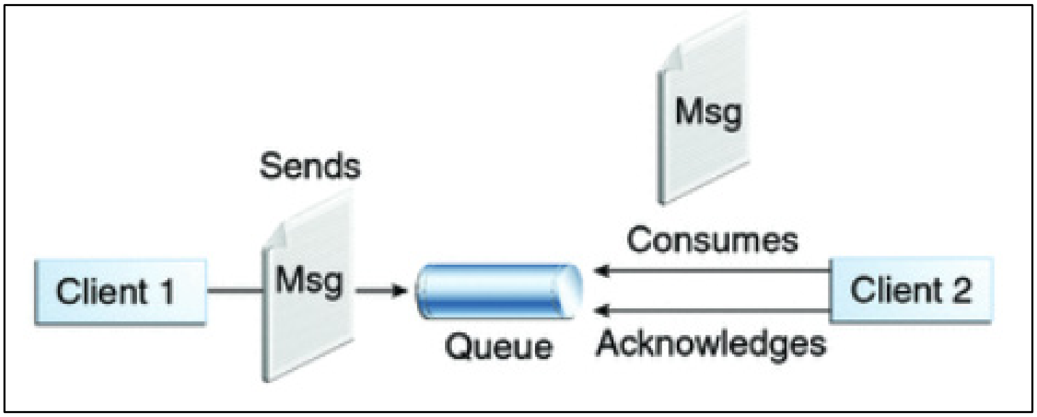


Figure 2. The one-to-one messaging model. Source: Jendrock et al. (2013a).

The Java EE messaging application sends the message asynchronously. In asynchronous communication, an application sends a message to a queue managed by messaging middleware and does not need to wait for a receiver's response because the receiver listens to the queue, which exists in messaging middleware and the receiver responds when it is ready (Richards et al., 2009). Asynchronous communication improves the performance of systems, as the application does not need to wait until it receives a response (Ferreira, 2013).

e. JMS messaging application components

The Java EE messaging application which we developed is a one-to-one JMS messaging application. According to JMS API specifications, the one-to-one model has the following components:

- A connection factory, which is an object obtained by a client that connects to a JMS messaging provider (Jendrock et al., 2013e).
- A connection is an object that is used to connect to JMS provider (Jendrock et al., 2013e).
- A session, which is an object that makes a messaging transaction work (Richards et al., 2009)
- A destination, which refers to both a queue and a consumer

In point-to-point JMS messaging, the destination refers to both the queue and the consumer because the queue holds messages and the consumer is the application that will receive the message (Jendrock et al., 2013e). Figure 3 describes the general components of a JMS application.

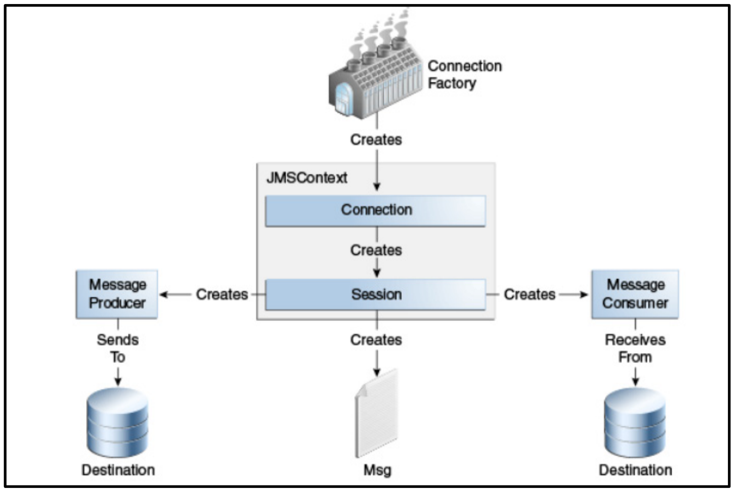


Figure 3. JMS messaging application components. Source: Jendrock et al. (2013e)

B. BONITA BPM

The other version of the messaging and persistence application developed for this study was created using Bonita BPM. Figure 4 illustrates the components of the Bonita BPM 6 platform, which is an open-source BPMS from Bonitasoft. Each component enables the development of process-centric applications. A main component of a standard BPMS platform is the execution engine, which runs and manages business processes (Harmon, 2014). Bonita Execution Engine (BEE), shown in Figure 4, consists of different components to run, configure, and manage process-centric applications (Allègre, 2013), the most relevant of which the following sections will explain.

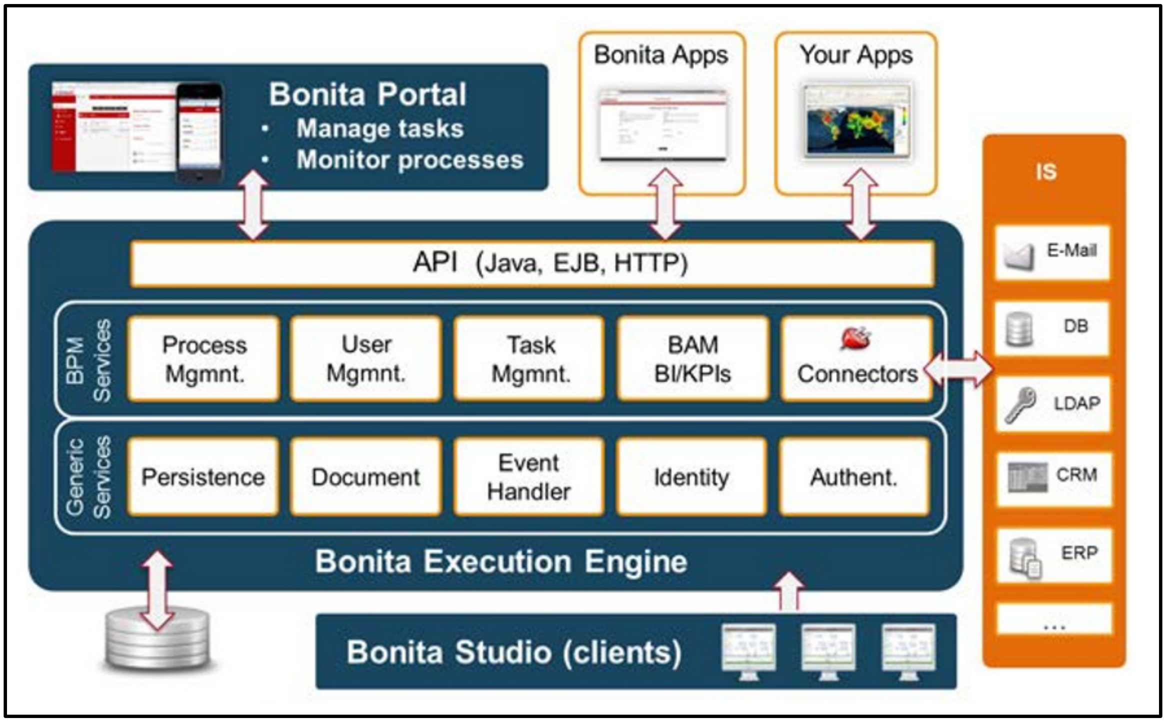
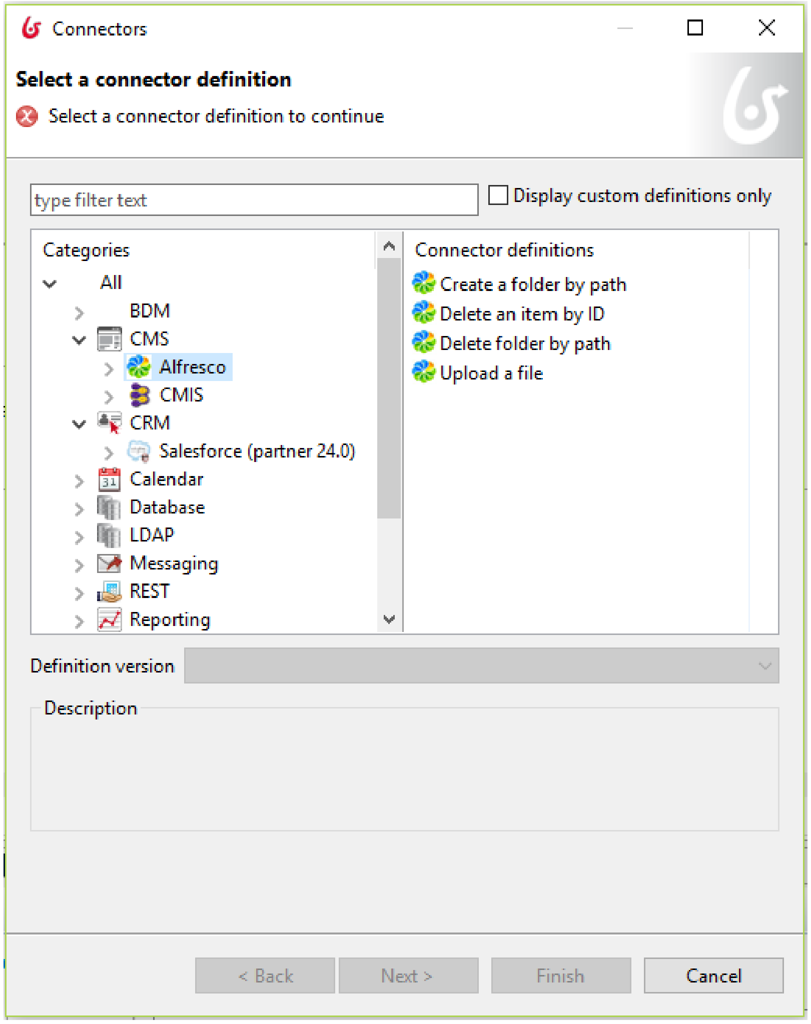


Figure 4. Bonita BPM 6 engine architecture. Source: Allègre (2013).

1. Connectors

Business applications must communicate with external systems for reading and writing information. Connectors are components in Bonita that are programmed and configured to facilitate the connection to external systems (Bonitasoft, n.d.b.). For example, an insurance approval process may require a document that has to be uploaded from a content management system (CMS); this process requires integration with the CMS system to acquire the document. To achieve this task, the Bonita BPM provides connectors that integrate the insurance approval process with the CMS. Bonita BPM offers many pre-programmed connectors to facilitate connectivity with external systems, such as for Customer Relationship Management (CRM) systems like Salesforce or CMSs such as Alfresco. Figure 5 illustrates the list of connectors that Bonita BPM provides.



Connectors

Select a connector definition

Select a connector definition to continue

Figure 5. Bonita BPM 7 offers different connectors to integrate external applications

4. Task Management

A task is an activity in the process that can be carried out by either humans or machines. Human tasks include the manual processes through which data is entered into business process application templates, while automated tasks of the Bonita BPM engine are performed in the background while the process is running. Bonita BPM engages other types of activities, such as the callback task, to call required subprocesses (Bonitasoft, n.d.e.).

5. Business Data Modeling

Business data is an essential part of enterprise systems in any organization. Business Data Modeling (BDM) is a component in Bonita BPM that allows developers to create and manage the data required for a business process in form of business objects to be persisted in a database (Bonitasoft, n.d.a.). To briefly illustrate how BDM works, consider a procurement order business process that starts with filling out a form that includes an item description and quantity. Here, the item description and quantity are data elements required to start the business process. In this case, a developer can create a business object that holds data elements—item description and quantity—by using BDM. The purpose of business objects is to hold the data required for a business process to be stored later in relational database systems and make it available to be accessed by any component in the business process itself or by any external business process.

6. Document

Business processes may require the production of certain documents (Bonitasoft, n.d.c.). Suppose there is a trainee nomination process that requires a copy of the trainee ID; this process requires attaching a file containing the image of the trainee card inserted through the file system or the content management system (CMS) (Bonitasoft, n.d.c.). Bonita BPM provides the ability to programmatically insert document files as variables of a string-type value that are stored in the process database.

BPMSs use BPMN standard to graphically represent the flow of business process. The next section gives a brief explanation about the BPMN standard.

7. Business Process Modeling in Bonita Studio

a. Modeling and Process Design Using the BPMN Standard

BPMN Business Process Management Notation describes business processes in a graphical format, depicted in Figure 6, that is easy to understand and analyze and helps developers create business-based applications (Stiehl, 2014).

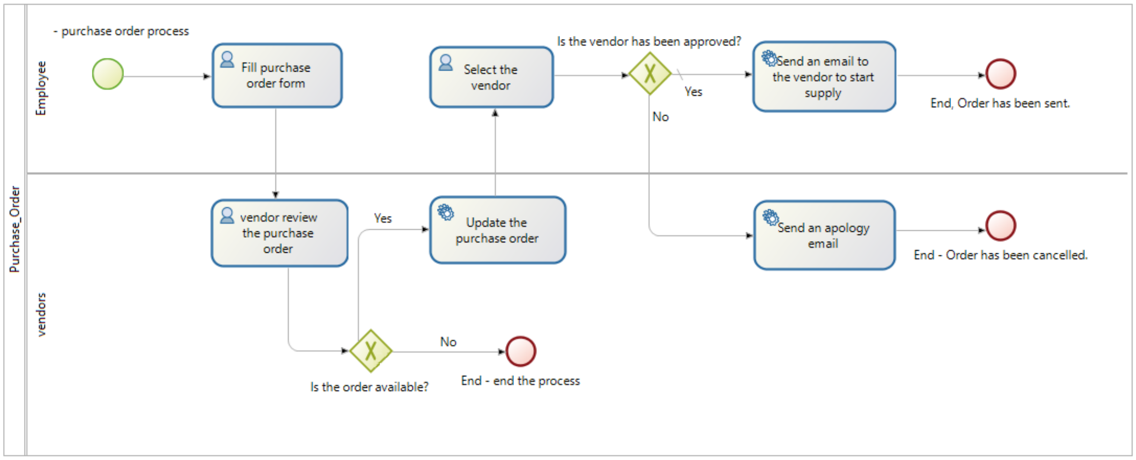
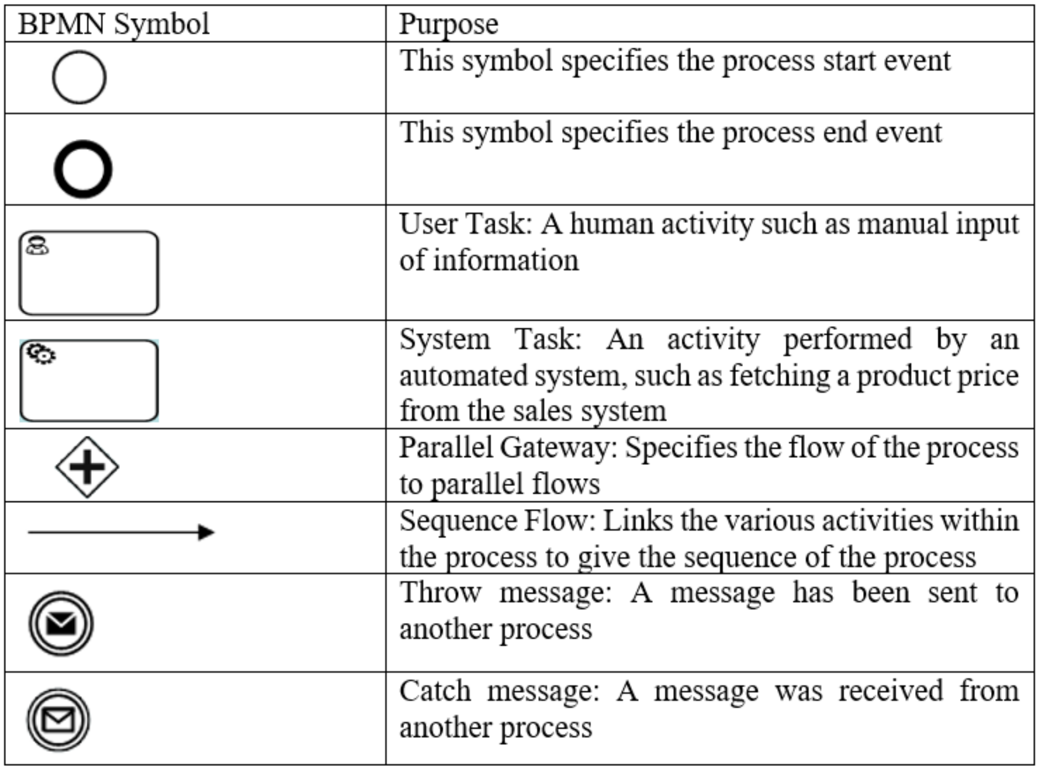

- purchase order process
Purchase_Order
Employee
vendors
Fill purchase order form
Select the vendor
Is the vendor has been approved?
Yes
No
Send an email to the vendor to start supply
End, Order has been sent.
vendor review the purchase order
Yes
Update the purchase order
Send an apology email
End - Order has been cancelled.
No
Is the order available?
End - end the process

Figure 6. Simple business process flow in BPMN standard.

The BPMN standard contains basic sets of graphical symbols that represent process flow, data, binding objects, lanes, and artifacts (Stiehl, 2014). Table 1 contains some of these BPMN symbols.

Table 1. Some of BPMN notations. Adapted from Open Management Group (2011).



BPMN Symbol	Purpose
○	This symbol specifies the process start event
◯	This symbol specifies the process end event
▭	User Task: A human activity such as manual input of information
▭	System Task: An activity performed by an automated system, such as fetching a product price from the sales system
⊕	Parallel Gateway: Specifies the flow of the process to parallel flows
→	Sequence Flow: Links the various activities within the process to give the sequence of the process
✉	Throw message: A message has been sent to another process
✉	Catch message: A message was received from another process

b. Bonita BPM Studio

Bonita BPM enables the creation of business process applications using BPMN standard through Bonita Studio, a business process application developer environment in Bonita BPM. Process components and workflow are managed by the Bonita BPM engine at the backend. The workflow in Figure 6 is an example of a business process that was created in Bonita Studio. The diagram displays different BPMN elements that describe the purchase order process flow. The process starts when the employee fills a purchase order and sends it out to several vendors. Each vendor receives the purchase order; if the order is in stock, the order will proceed to the next step, and if not, the process ends. If the order is in stock, an automated system will update the order and send it back to the employee. The employee receives these updates from different vendors and can manually choose which vendor is approved. Based on this decision, an automatic email will be sent to notify the vendor to start supply, or the vendor receives apology email with a rejection.

III. JAVA EE AND BONITA BPM APPLICATIONS

This chapter includes the Java EE, Bonita BPM persistence and messaging application. First section includes the persistence applications for Java EE and Bonita BPM. It demonstrates the development steps for persistence applications. The second section includes the messaging applications for Java EE and Bonita BPM. It demonstrates the development steps for messaging application.

A. PERSISTENCE APPLICATIONS

This section includes the development steps that we accomplished to develop the persistence application.

1. Java EE Persistence Application

We first developed a Java EE application to demonstrate the Java EE persistence feature. The application is a simple database web application to store student thesis data. This application could be deployed and run on a server. It is composed of three main components: the view component, which I will call the user interface; the controller component, which I will call the business logic; and the model component, which I will call the data resource. Figure 7 illustrates these major components of the thesis database application.

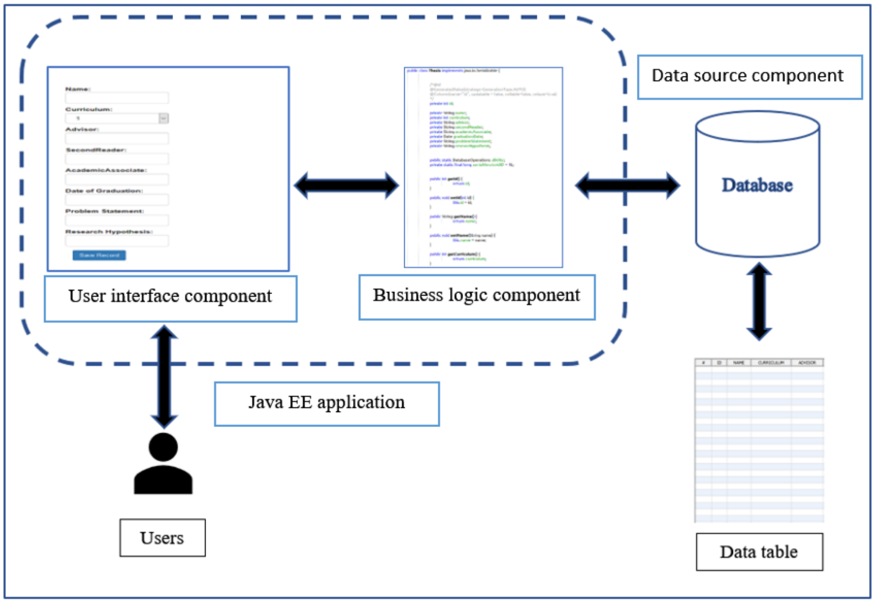


Figure 7. A general overview of the major components of thesis application

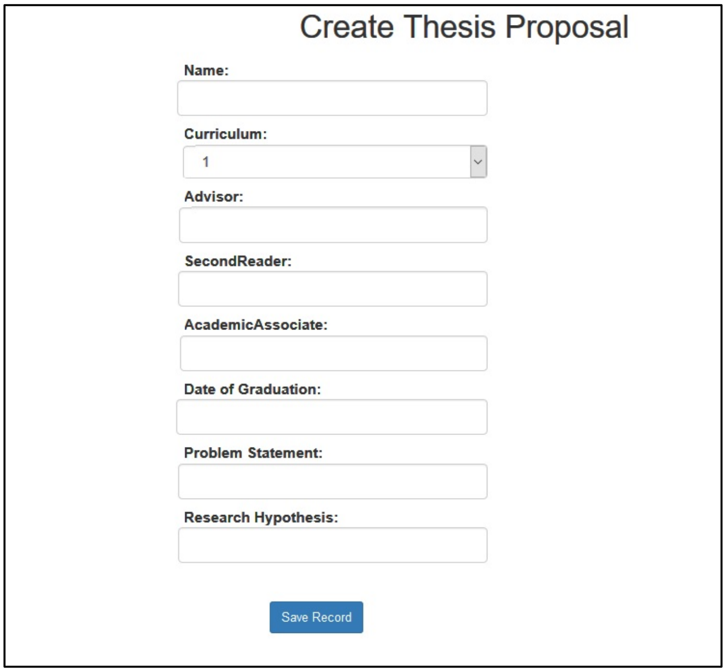
To build each component, we used the Java EE platform. Table 2 lists the Java EE technologies used to create our application.

Table 2. Java EE technologies, which is used to develop the persistence application

Application component	Java EE technology
User interface	JSF technology
Business logic	JPA Hibernate Plain Old Java Object (POJO) class.
Data source	JavaDB

a. User Interface

The user interface is a web page that contains eight data-entry fields, used to enter thesis information, and one button to save data in a database. We used JSF technology to design the application's web page. Figure 8 shows the application's single user interface.



Create Thesis Proposal

Name:

Curriculum: 1

Advisor:

SecondReader:

AcademicAssociate:

Date of Graduation:

Problem Statement:

Research Hypothesis:

Figure 8. Java EE persistence web page

b. Business Logic

The business logic component consists of Java code to handle and transfer the data between the web interface and the database. The business logic component contains different technologies we implemented to demonstrate Java EE persistence. The data source component is the database we used to store thesis data.

For the business logic, we used JPA API—specifically, Hibernate technology. To explain the role and the purpose Hibernate technology, we have to explain what is happening to the data when a user starts entering thesis information through the application's interface until it is persisted or stored in the database.

To store data into the database, the user enters Name, Curriculum, Thesis advisor, etc. in the designated input fields in the web page, which means the data will be entered through the user interface component. In our application, therefore, the data will be stored temporarily in a thesis object instantiated from the "thesis" Java class; however, the thesis is a relational table in the database. Java classes and relational data tables are totally different structures: Java classes consist of fields, properties, and methods and are based on an Object-Oriented Programming (OOP) model, whereas relational tables consist of columns and relations between these columns.

The difference between OOP and Relational models was previously an issue that hardened data access from OOP models to a relational model (Kieth, Schincariol, Nardone, 2018). However, Java JPA API standard solves the mismatch issue between OOP and relational models by using a mechanism called object-relational mapping (ORM) (Kieth, Schincariol, Nardone, 2018). The ORM approach allows an object-based system to store data in relational databases (O'Neil, 2008).

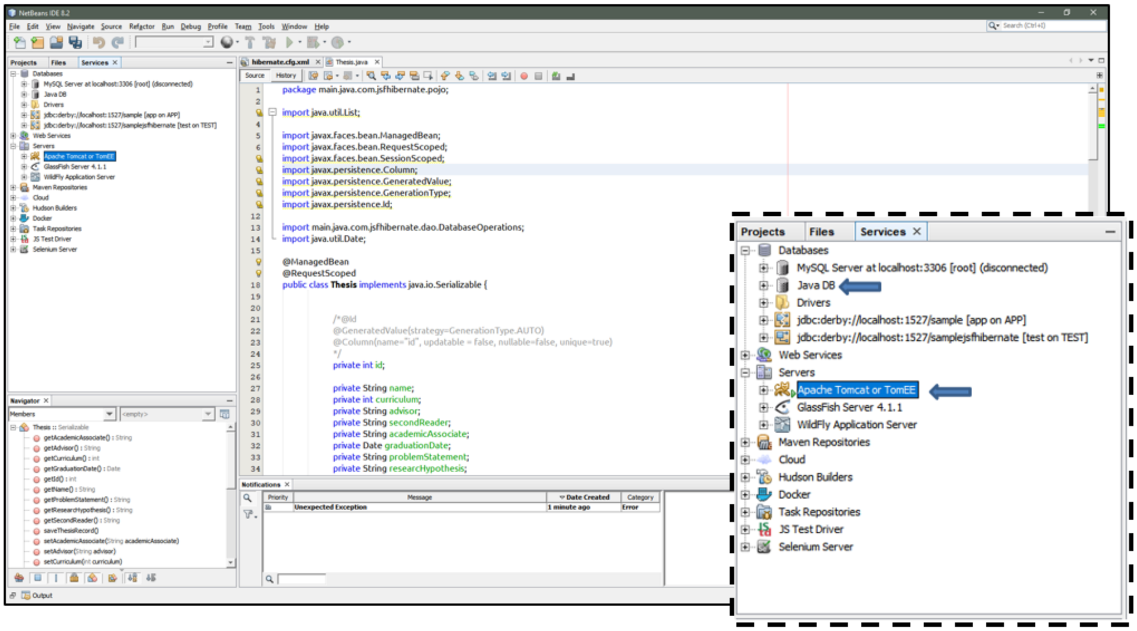
Hibernate is an ORM software compatible with the JPA standard. It uses ORM methodology to persist data in relational database. In our application, we use Hibernate to map the thesis Java class fields to thesis table columns in JavaDB databse.

c. Data Source

JavaDB is an open-source lightweight relational database used to develop and test database applications. Developers prefer to use lightweight databases to test their applications before final production. The other reason for choosing JavaDB is that it has enough functions to serve the purpose of our study.

d. Application Development Tool

We used the Netbeans tool to develop our application. Netbeans is an Integrated Development Environment (IDE) used to create, test, and run Java applications (Netbeans, 2018). Netbeans IDE offers pre-installed and preconfigured development tools such as Apache Tomcat and JavaDB. Apache Tomcat is a type of software called a web server used to run and deploy Java web-based application (Apache, 2018). We used Apache Tomcat to run and deploy our application. Figure 9 shows the Apache tomcat application server and JavaDB configured inside Netbeans IDE.



```java
package main.java.com.jsfhibernate.pojo;

import java.util.List;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.bean.SessionScoped;
import javax.persistence.Column;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

import main.java.com.jsfhibernate.dao.DatabaseOperations;
import java.util.Date;

@ManagedBean
@RequestScoped
public class Thesis implements java.io.Serializable {


        /*@Id
        @GeneratedValue(strategy=GenerationType.AUTO)
        @Column(name="id", updatable = false, nullable=false, unique=true)
        */
        private int id;

        private String name;
        private int curriculum;
        private String advisor;
        private String secondReader;
        private String academicAssociate;
        private Date graduationDate;
        private String problemStatement;
        private String researcHypothesis;
```

Notifications

Priority	Message	Date Created	Category
	Unexpected Exception	1 minute ago	Error

Figure 9. Netbeans IDE used to develop the Java EE persistence application

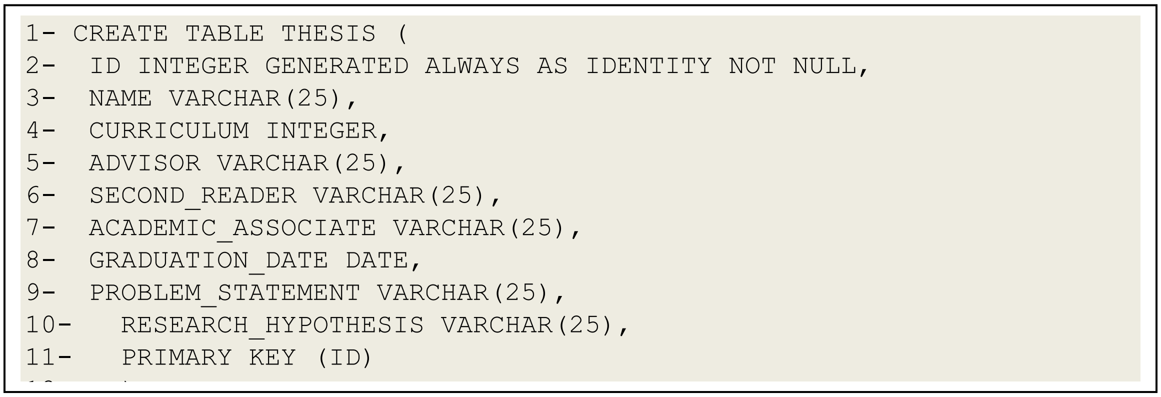
e. *Developing the Java Persistence Application*

(1) Create a database

We started developing our application by creating a database using JavaDB in the Netbeans development environment. We created a database and named it samplejsfhibernate, then we created a schema named THESIS containing the data table that we will create in the next step.

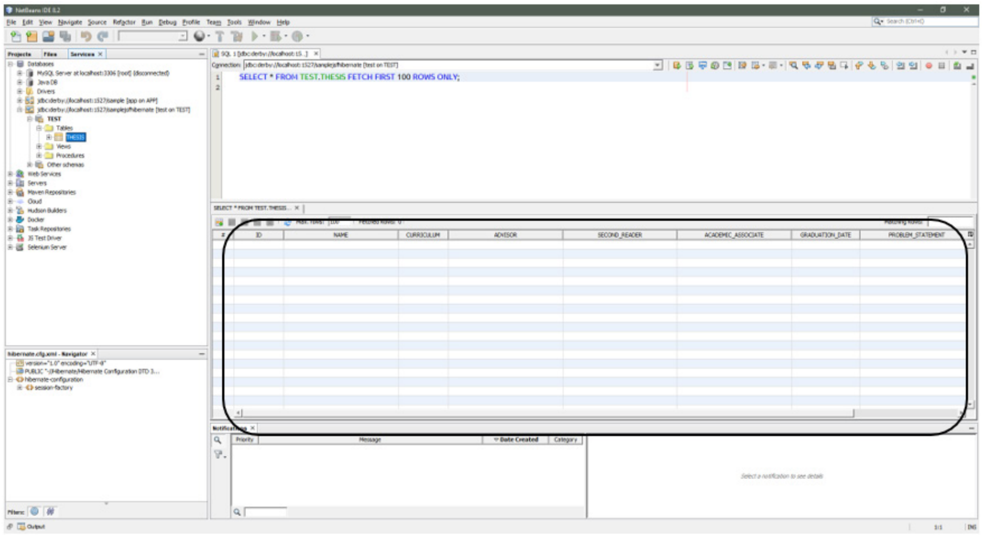
(2) Create thesis table in JavaDB relational database system

The next step was to create a table in the database in which the data will be stored (persisted). Database systems use a query language called Structured Query Language (SQL) to manage and manipulate data. In the code shown in Figure 10, in line 1, we used the `Create Table` command to create a table in the database; we named this table `THESIS`. Within this table, we created several columns, each column containing a specific name and type of data. For example, in line 3, the column `NAME` is the column that will store the name data, which will be entered through the application end-user interface. We defined the NAME data type as a 25-character VARCHAR data type. In the same way, we created the remaining columns and associated them with names and data types. Figure 11 shows the resulting data table in Netbeans IDE.



```
1- CREATE TABLE THESIS (
2-  ID INTEGER GENERATED ALWAYS AS IDENTITY NOT NULL,
3-  NAME VARCHAR(25),
4-  CURRICULUM INTEGER,
5-  ADVISOR VARCHAR(25),
6-  SECOND_READER VARCHAR(25),
7-  ACADEMIC_ASSOCIATE VARCHAR(25),
8-  GRADUATION_DATE DATE,
9-  PROBLEM_STATEMENT VARCHAR(25),
10-   RESEARCH_HYPOTHESIS VARCHAR(25),
11-   PRIMARY KEY (ID)
12-   )
```

Figure 10. SQL code used to create Thesis table in JavaDB database


NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Files Services
Databases
MySQL Server at localhost:3306 [root] (disconnected)
Java DB
Drivers
jdbc:derby://localhost:1527/sample [app on APP]
jdbc:derby://localhost:1527/samplejsfhibernate [test on TEST]
TEST
Tables
THESIS
Views
Procedures
Other schemas
Web Services
Servers
Maven Repositories
Cloud
Hudson Builders
Docker
Task Repositories
JS Test Driver
Selenium Server
hibernate.cfg.xml - Navigator
Connection: jdbc:derby://localhost:1527/samplejsfhibernate [test on TEST]
Notifications
Priority
Message
Date Created
Category
Output

```
SELECT * FROM TEST.THESIS FETCH FIRST 100 ROWS ONLY;
```

#	ID	NAME	CURRICULUM	ADVISOR	SECOND_READER	ACADEMIC_ASSOCIATE	GRADUATION_DATE	PROBLEM_STATEMENT

Figure 11. Data table “Thesis” as it appears in the development environment

(3) Create Java Class

We created a Java class named Thesis. We defined the class properties such as “id” and “name” (line 4 and 5) in Figure 12. The class properties store (temporarily) the thesis information entered through the web page. To make the class properties accessible to the JSF, we used Java notations `@MangedBeans` and `@RequstScoped.` This means we wired the input text elements in the web page to the Thesis class properties. We defined the getters for each property.

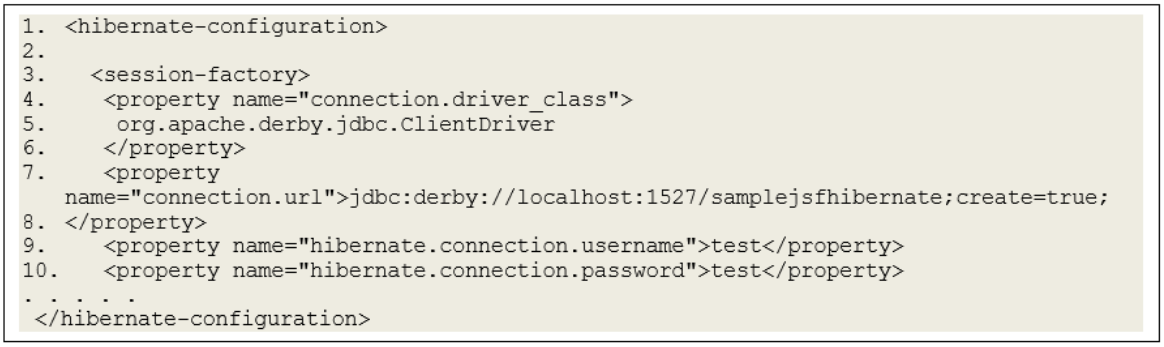
```
@ManagedBean
@RequestScoped
public class Thesis implements java.io.Serializable {
 private int id;
 private String name;
 ……
 public int getId() {
  return id;
 }
 public void setId(int id) {
  this.id = id;
 }
 …
}
```

Figure 12. “Thesis” Java class

(4) Create Hibernate configuration file

Hibernate uses an XML file called a Hibernate configuration file to know what database driver will be used to connect to the database and to know the name and the location of the database. JavaDB uses a database driver called `ClientDriver`. In the line number 4 in Figure 13, we specified the path for the JavaDB driver.

Hibernate also has to know the location of the database path. In line 7, we defined the path of the database location associated with its name, `samplejsfhibernate.` In lines 9 and 10, Hibernate uses the username and password to connect to the database. However, there are many other configurations that could be set up in the Hibernate configuration file based on the application’s requirements.

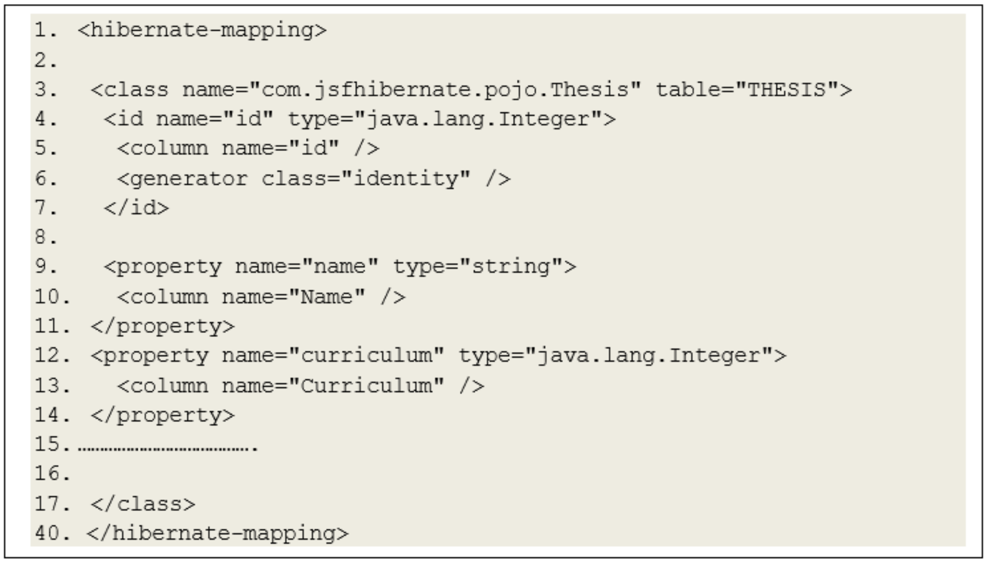


```
<hibernate-configuration>

  <session-factory>
   <property name="connection.driver_class">
    org.apache.derby.jdbc.ClientDriver
   </property>
   <property
name="connection.url">jdbc:derby://localhost:1527/samplejsfhibernate;create=true;
</property>
   <property name="hibernate.connection.username">test</property>
   <property name="hibernate.connection.password">test</property>
. . . . .
 </hibernate-configuration>
```

Figure 13. Hibernate configuration file, which contains paths for database driver and database location

(5) Create Hibernate Mapping XML File

Hibernate uses an XML file called Hibernate mapping file to map Java class fields to table columns in the database. In this file, we configure which Java class field is associated with which column in the data table that we created in the database. For example, line number 9 includes the Java class field called "name" that will hold values of string type `<property name="name" type="string">`, which is mapped to a column in the data table called "Name" `<column name="Name" />`. Mapping requires that the Java class field data type should be the same as the column data type. We can see in line 9 and 10 the data table column "Name," which we defined earlier in SQL script as containing the VARCHAR data type, which means the "Name" column can hold data of varying characters. In Java class, we defined the "name" field as a string data type, which means "name" can hold a character-type data value. String and VARCHAR data types can be matched together because they can hold the same data types and values. In the same way, we map the rest of the Java class fields to their relevant columns in the data table. Figure 14 shows the sample code of the Hibernate mapping file in the Java EE persistence application.

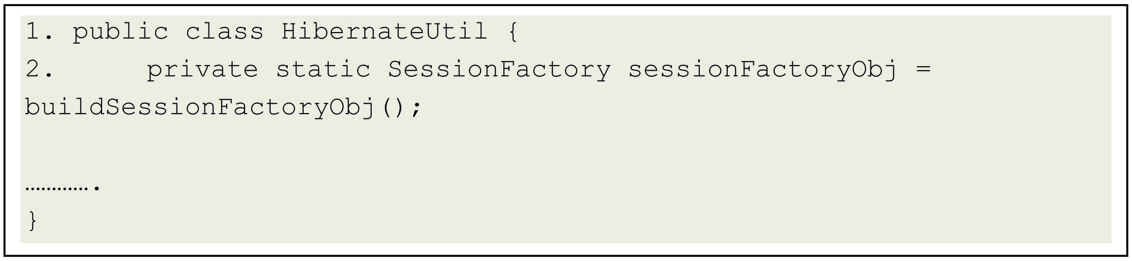


```
<hibernate-mapping>

 <class name="com.jsfhibernate.pojo.Thesis" table="THESIS">
  <id name="id" type="java.lang.Integer">
   <column name="id" />
   <generator class="identity" />
  </id>

  <property name="name" type="string">
   <column name="Name" />
 </property>
 <property name="curriculum" type="java.lang.Integer">
   <column name="Curriculum" />
 </property>
……………………………….

 </class>
</hibernate-mapping>
```

Figure 14.   Sample code of Hibernate mapping file

(6)   Create Hibernate utility class

The purpose of this class is to create a session factory, which allows us to create a session object. We use session object methods to persist the data into the database. In Hibernate utility class, we created a session factory object called “sessionFactoryObj” Figure 15.



```
public class HibernateUtil {
     private static SessionFactory sessionFactoryObj =
buildSessionFactoryObj();

………….
}
```

Figure 15.   Sample code of Hibernate utility class

(7)   Create a database operation class

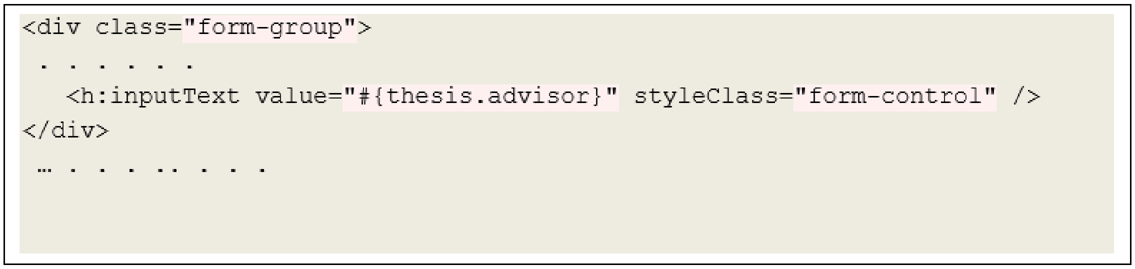
To enable Hibernate to create, read, delete or update the mapped data elements into the database, it requires a session object. The session object offers a method called `save()`, to persist the data in the database. We used the Hibernate utility class to create the session object. Before we use the session's `save()` method, it is required to create a transaction object. We created a transaction object named "transObj" and then created a session object named "sessionObj.". Figure 16 shows a sample code of database operation class.

```
public class DatabaseOperations {

        private static Transaction transObj;
        private static Session sessionObj =
HibernateUtil.getSessionFactory().openSession();
        // Method To Add New Thesis Details In Database
        public void addThesisInDb(Thesis thesisObj) {
                try {
                        transObj = sessionObj.beginTransaction();
                        sessionObj.save(thesisObj);
        ……….. }
        }
```

Figure 16. Sample code of database operation class

(8) Create a user interface

We used JSF technology to create and design the interface. It is a web page contains eight input text fields and one button. All these elements are wired with Java class properties. We used JSF <h: inputText > tag to create input text on the web page. The value the user will enter in the input text will be passed to the specified Java class property by using # tag. Figure 17 shows a sample JSF code we created for the application's web page. The user inputs in the "advisor" field will be passed to the "advisor" property.



```
<div class="form-group">
 . . . . . .
   <h:inputText value="#{thesis.advisor}" styleClass="form-control" />
</div>
 … . . . . .. . . .
```

Figure 17. Sample code of web page code in JSF

2. Bonita BPM Persistence Application

We used a business process application called Expense Report created by Bonitasoft to explore how Bonita BPM persists data (Bonitasoft, n.d.f.). The business process sequence that the application performs consists of the following steps:

1. A user fills out a form called an expense report by entering report summary, item names, and item expense information into designated fields. A user can add more than one item. Then he submits the form for approval by his manager.

2. The manager receives the form and can accept or reject the request.

3. If the manager approves the form, then the process ends.

4. If the manager rejects the form, it will be returned to the employee.

The required steps to create business process application in Bonita BPM are based on the following steps (Ozill, 2015):

1. Model the business process diagram by using BPMN.

2. Define BDM objects.

3. Set up contracts, BDM instances, operations, groovy scripts, connectors,…

4. Design interfaces.

These steps are not all the steps required to develop a business process application, as the study of this application focuses on data persistence. The steps used to create the application are described in the sections that follow.

(1) Model business process with BPMN standard

The business process modeling involves representing the process flow by using BPMN notation. Figure 18 shows the BPMN diagram, which describes the flow of expense reports. We numbered each BPMN element in the diagram to explain its function. A developer created a BPMN element called Pool (1), which contains two lanes (2), an Employee lane and a Manager lane. The business process starts with an event (3), when an employee fills out the form and submits the form to his manager. A transition line (4) indicates the next step in the process flow, (5), a human activity, which is when the manger reviews the report. Then an XOR gate (6) indicates that a decision has to take a place. If the manger accepts the form, then the process will end. If the manager rejects the form, then the request will be sent back to the employee.

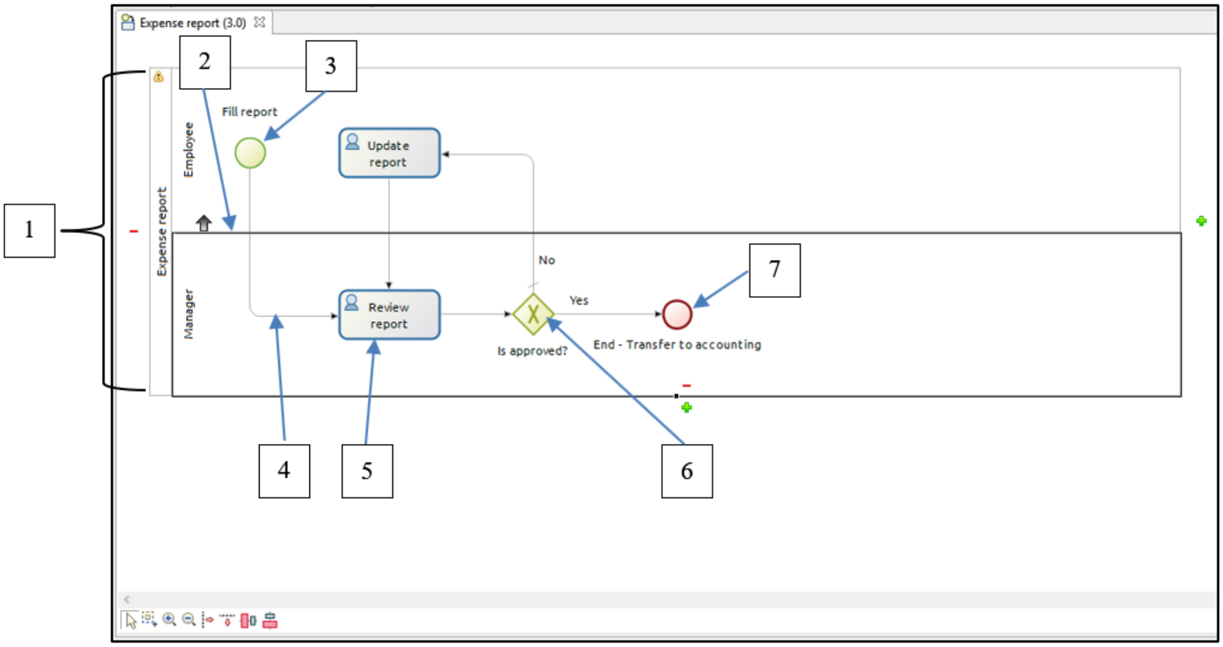
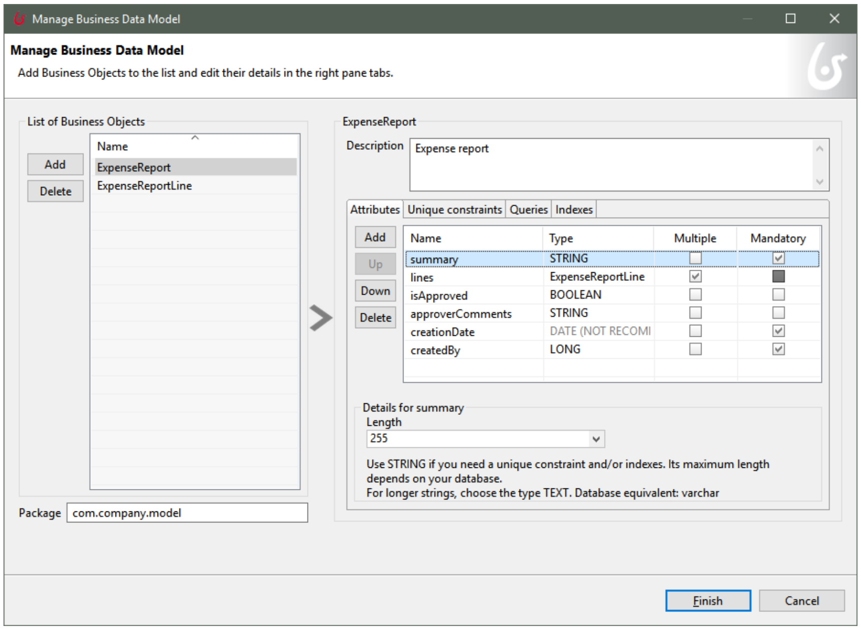


Figure 18. Expense Report business process flow BPMN diagram

Here, the BPMN diagram is still only a graphical representation and not ready to be executed by the Bonita BPM engine. The next steps show how the BDM was configured.

(2) Define BDM objects

Bonita BPM 7 provides BDM, which is a structure to manage business objects (Bonitasoft, n.d.a.). Business objects are elements of data required for a business process. BDM defines business objects as Java classes to persist data in a relational database (Bonitasoft, n.d.a.). Business objects' names have to be written as formal Java class names, which do not accept spaces (Bonitasoft, n.d.a.). It is a Java programming rule to write Java class names without spaces because they cause programming errors. In the expense report application, ExpenseReport and ExpenseReportLine business objects were created in BDM. They consist of attributes to persist data. The attributes were defined with their data types. Figure 19 shows the defined business objects with their attributes.



Manage Business Data Model

Add Business Objects to the list and edit their details in the right pane tabs.

List of Business Objects

Name
ExpenseReport
ExpenseReportLine

Package: com.company.model

ExpenseReport

Description: Expense report

Name	Type	Multiple	Mandatory
summary	STRING	□	☑
lines	ExpenseReportLine	☑	■
isApproved	BOOLEAN	□	□
approverComments	STRING	□	□
creationDate	DATE (NOT RECOMI	□	☑
createdBy	LONG	□	☑

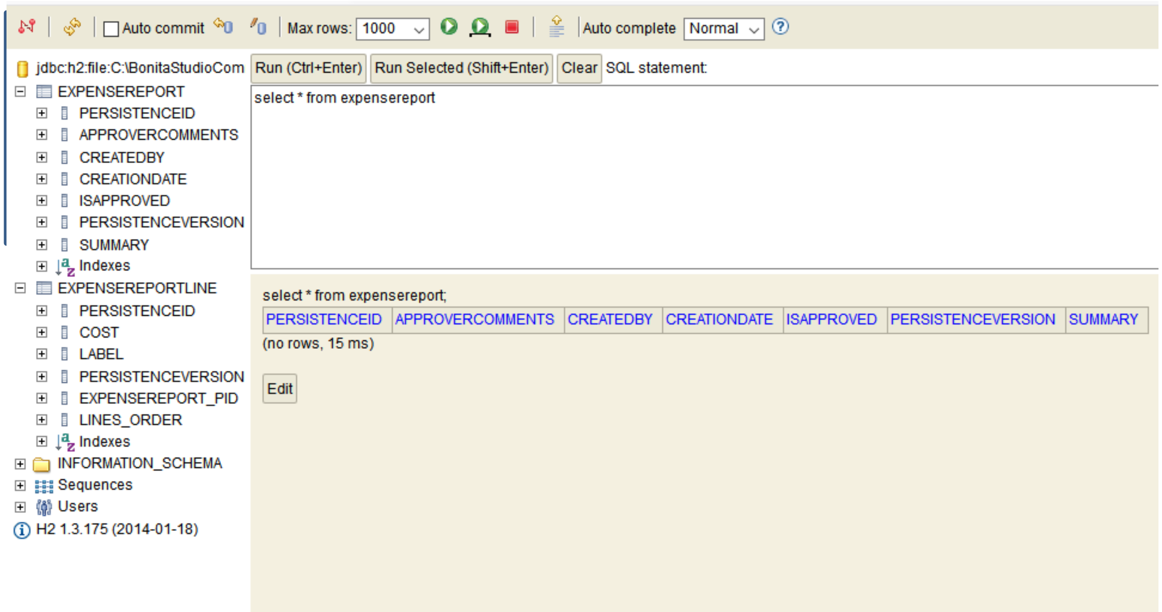
Details for summary

Length: 255

Use STRING if you need a unique constraint and/or indexes. Its maximum length depends on your database.
For longer strings, choose the type TEXT. Database equivalent: varchar

Figure 19. Managing business objects in BDM

After the business objects had been defined with their attributes, the BDM generated two data tables, named EXPENSEREPORT and EXPENSEREPORTLINE, in an embedded relational database called H2. Figure 20 shows the generated data tables in the H2 database. BDM converts the defined business objects with their attributes to Java classes associated with their properties, which will be used to persist data later in a relational database. Unlike in the Java application, BDM automatically generates the data table and automatically maps each Java class property onto its proper column in the data table. Bonita BPM 7 uses Hibernate as an ORM provider to map business object attributes to a relational database (Ozil, 2015).



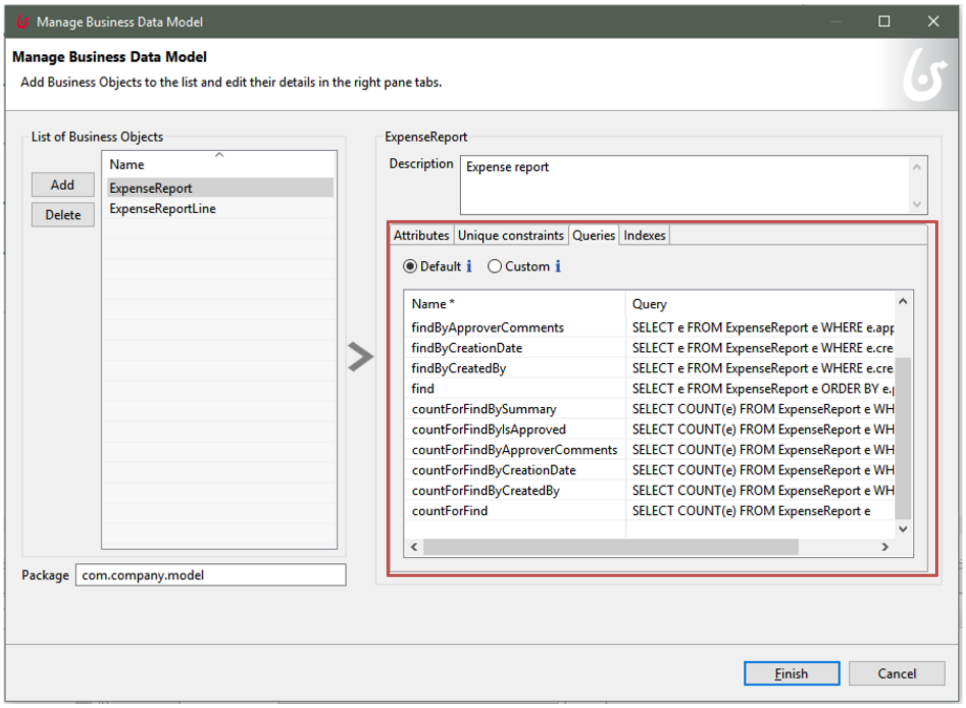
```
jdbc:h2:file:C:\BonitaStudioCom
⊟ EXPENSEREPORT
    ⊞ PERSISTENCEID
    ⊞ APPROVERCOMMENTS
    ⊞ CREATEDBY
    ⊞ CREATIONDATE
    ⊞ ISAPPROVED
    ⊞ PERSISTENCEVERSION
    ⊞ SUMMARY
    ⊞ Indexes
⊟ EXPENSEREPORTLINE
    ⊞ PERSISTENCEID
    ⊞ COST
    ⊞ LABEL
    ⊞ PERSISTENCEVERSION
    ⊞ EXPENSEREPORT_PID
    ⊞ LINES_ORDER
    ⊞ Indexes
⊞ INFORMATION_SCHEMA
⊞ Sequences
⊞ Users
H2 1.3.175 (2014-01-18)
```

```
select * from expensereport
```

```
select * from expensereport;
PERSISTENCEID  APPROVERCOMMENTS  CREATEDBY  CREATIONDATE  ISAPPROVED  PERSISTENCEVERSION  SUMMARY
(no rows, 15 ms)
```

Figure 20. BDM generated an EXPENSEREPORT and EXPENSEREPORTLINE in H2 database.

BDM also automatically generates a Java Persistence Query Language (JPQL) code for each business object attribute. JPQL is required by Hibernate to persist the data. Figure n shows the generated JPQL code for each business object attribute.



Package: com.company.model

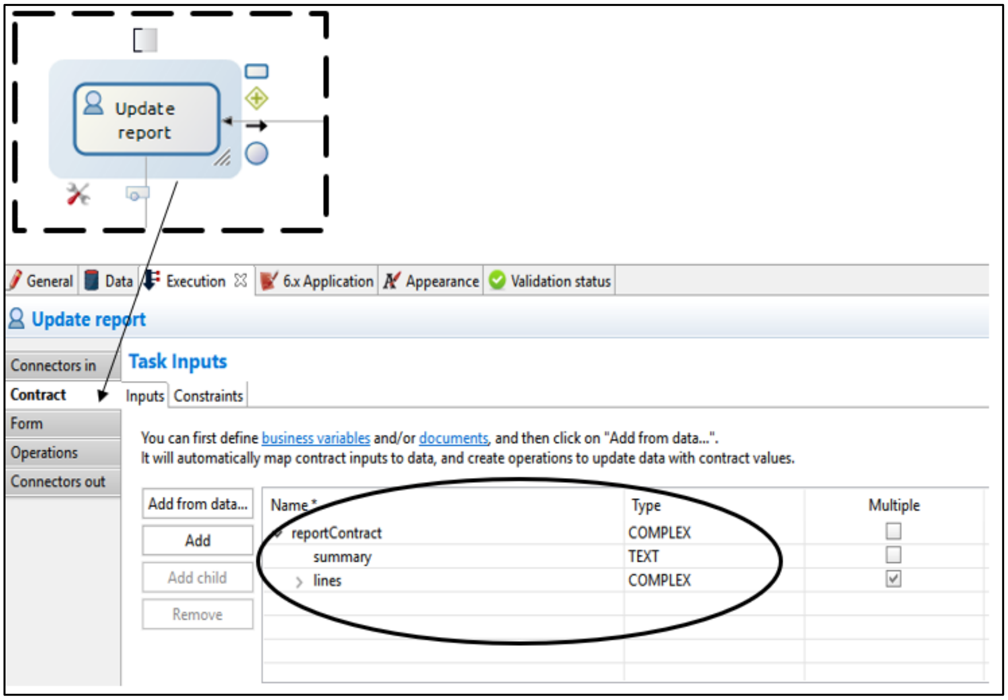
Description: Expense report

Name *	Query
findByApproverComments	SELECT e FROM ExpenseReport e WHERE e.app
findByCreationDate	SELECT e FROM ExpenseReport e WHERE e.cre
findByCreatedBy	SELECT e FROM ExpenseReport e WHERE e.cre
find	SELECT e FROM ExpenseReport e ORDER BY e.
countForFindBySummary	SELECT COUNT(e) FROM ExpenseReport e WH
countForFindByIsApproved	SELECT COUNT(e) FROM ExpenseReport e WH
countForFindByApproverComments	SELECT COUNT(e) FROM ExpenseReport e WH
countForFindByCreationDate	SELECT COUNT(e) FROM ExpenseReport e WH
countForFindByCreatedBy	SELECT COUNT(e) FROM ExpenseReport e WH
countForFind	SELECT COUNT(e) FROM ExpenseReport e

Figure 21. Hibernate ORM generated JPQL code in BDM

(3) Setup the contracts

Contracts is a concept provided by Bonitasoft that defines the required valid data inputs for each activity in the business process flow (Ozil, 2015). A contract includes one or more input validation parameters associated with defined activities in a business process. Figure 22 shows a contract named “reportContract”, which includes defined input parameters: a summary and lines required for the “Update report” activity. We can see in Figure 22 that the “summary” input parameter is defined as a TEXT data type, which means a user has to enter a text value in the “summary” input field in the user interface associated with the activity.



You can first define business variables and/or documents, and then click on "Add from data...".
It will automatically map contract inputs to data, and create operations to update data with contract values.

Name *	Type	Multiple
reportContract	COMPLEX	□
summary	TEXT	□
lines	COMPLEX	☑

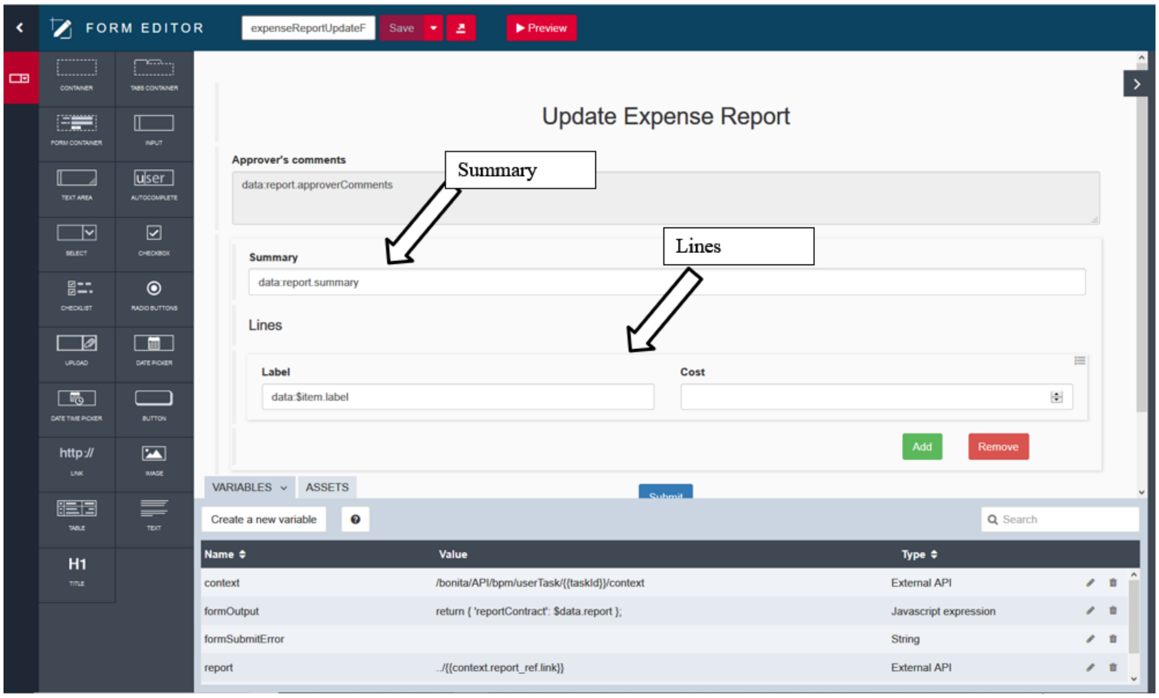
Figure 22. The defined contract for Update report activity

(4) Interface Design

Bonita BPM provides a user interface designer called UI designer, which allows developers to create and configure user interfaces. The UI designer uses AngularJS, which is a technology that allows the development of user interfaces for web-based applications (Ozil, 2015). UI designer offers many ready-to-use interface components to be used create web interface. Figure 23 shows the UI designer in Bonita BPM. In Figure 23, we can see the input text fields “Summary” and “Lines,” which will be linked to contract parameters “Summary” and “Lines.”

(5) Design the user interface

However, the UI designer allows developers to create a user interface in a pure coding environment called Bootstrap, shown in Figure 23 (Ozil, 2015).



Update Expense Report

Approver's comments: data:report.approverComments

Summary: data:report.summary

Lines

Label: data:$item.label

Cost:

Name	Value	Type
context	/bonita/API/bpm/userTask/{{taskId}}/context	External API
formOutput	return { 'reportContract': $data.report };	Javascript expression
formSubmitError		String
report	../{{context.report_ref.link}}	External API

Figure 23. User interface designer in Bonita BPM

B. MESSAGING APPLICATIONS

This section describes the process to create messaging applications in Java EE and Bonita BPM.

1. Java Messaging Application

We first developed a Java EE application to demonstrate the Java EE messaging feature. The application is a simple point-to-point messaging web-based application that uses JMS API to send a text message to a queue, then a receiver application reads the message from the queue and acknowledges it. Figure 24 shows a high-level overview of the messaging application. We used ActiveMQ to manage and deliver the message—

specifically, the version of ActiveMQ that comes inside the application server called Wildfly. In this application, the Java class we named “requstBean” is used to send the message. We created another Java class named “SimpleMessageBean”, which is a message-driven bean that listens to the queue, reads the message from the queue, and then acknowledges that the message is received.

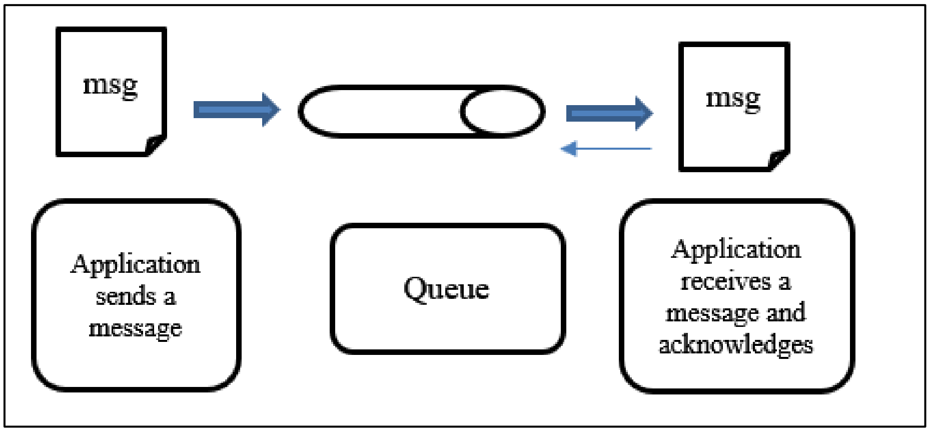

msg
msg
Application sends a message
Queue
Application receives a message and acknowledges

Figure 24. General overview of the Java EE messaging application. Adapted from (Jendrock et al., 2013a).

Developing the Java EE messaging application

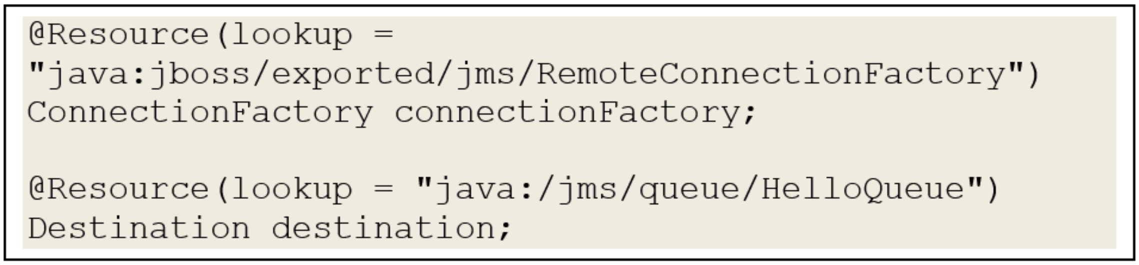
The required steps to develop this application were as follows:

(1) Create queues in ActiveMQ

We created a queue in ActiveMQ named `HelloQueue.` We added another required queue to establish the connection.

(2) Create a managed bean Java class.

In this class, we used JMS API to establish the connection to the queue, create the session, and create the message producer. In this class, we used the @Resource notations to reference ConnectionFactory to obtain a new connection factory object in line 1. In line 2, we referenced the destination of the message, which is the queue that we created in ActiveMQ:



```
@Resource(lookup =
"java:jboss/exported/jms/RemoteConnectionFactory")
ConnectionFactory connectionFactory;

@Resource(lookup = "java:/jms/queue/HelloQueue")
Destination destination;
```

Figure 25. Sample code of Java managed bean class

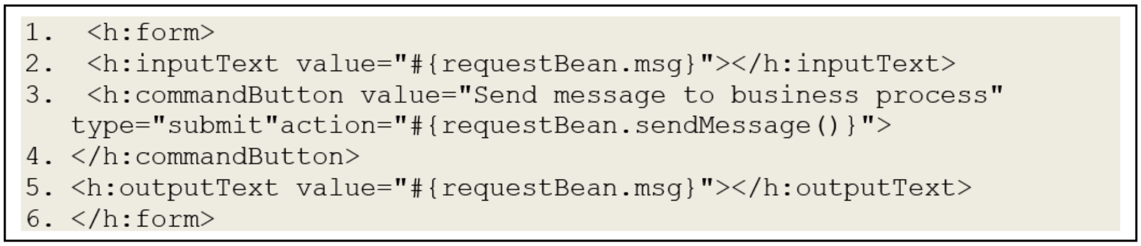
We defined the Connection Factory, queue connection, Session and Message Producer. To create a new queue connection object obtained through the connection factory object, we created a queue session configured to be in acknowledge mode, so the receiver will acknowledge that the message is delivered. Then we created the message producer object, which sends the message. See the code in appendix B.

(3) Create a simple message-driven bean

The message-driven bean is a Java class that listens to the queue and reads any message in the queue. In this class, we defined the destination queue and a method that implies that on receiving a message, the message-driven bean will call another process or application. See the code in appendix B.

(4) Create web page

We created the web page by using JSF. This page contains an input text box and a button to submit the message to the managed bean. Figure 26 shows sample code of web page in JSF.



```
 <h:form>
 <h:inputText value="#{requestBean.msg}"></h:inputText>
 <h:commandButton value="Send message to business process"
type="submit"action="#{requestBean.sendMessage()}">
</h:commandButton>
<h:outputText value="#{requestBean.msg}"></h:outputText>
</h:form>
```

Figure 26. Sample code of Java EE messaging application in JSF

2. Bonita BPM Messaging Application

We also developed a messaging application using Bonita BPM. The application is composed of two pools; each pool has a business process. When the first business process is executed, it sends a message the second business process to be started. We created this application by dragging and dropping Pool, start event, transitions, human activities, throw message, and catch message notations into the process diagram workspace in Bonita BPM workspace. Figure 27 shows the business process flow diagram in BPMN. The first process includes the human activity named Step1, which sends a message through the Message1 element. Message1 is a BPMN throw message element that sends a message to another process element in another Pool. We configured the Message1 element to specify the targeted Pool and the targeted business process element in the same Pool. Figure 28 shows the Message1 configuration window, where we defined that the targeted Pool is Pool3 and the targeted element is Step2.

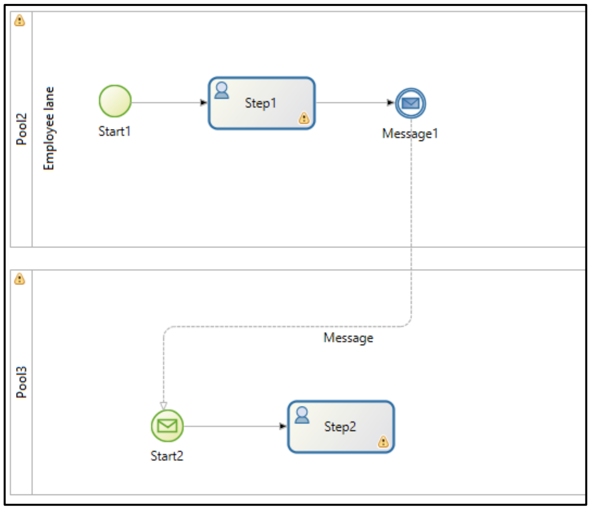
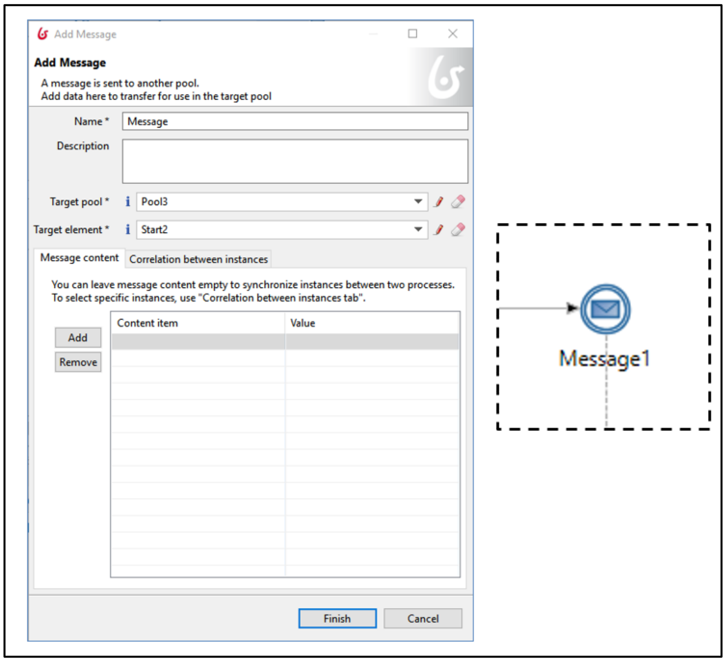


Figure 27. The BPMN diagram of the Bonita BPM messaging application


Add Message

Add Message

A message is sent to another pool.
Add data here to transfer for use in the target pool

Name * Message

Description

Target pool * Pool3

Target element * Start2

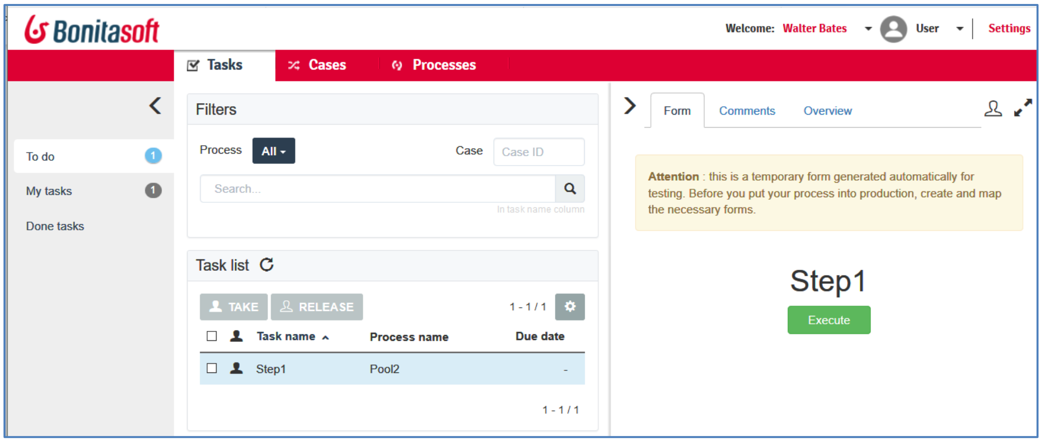
Message content | Correlation between instances

You can leave message content empty to synchronize instances between two processes.
To select specific instances, use "Correlation between instances tab".

Content item	Value

Figure 28. The configuration window of the throw message element

We ran the application and executed the first business process. Figure 29 shows that the Bonita Portal contains the first business process involving Step 1 activity, where it is listed on the task list. We executed the Step 1 activity, which sends the message to start the second business process.

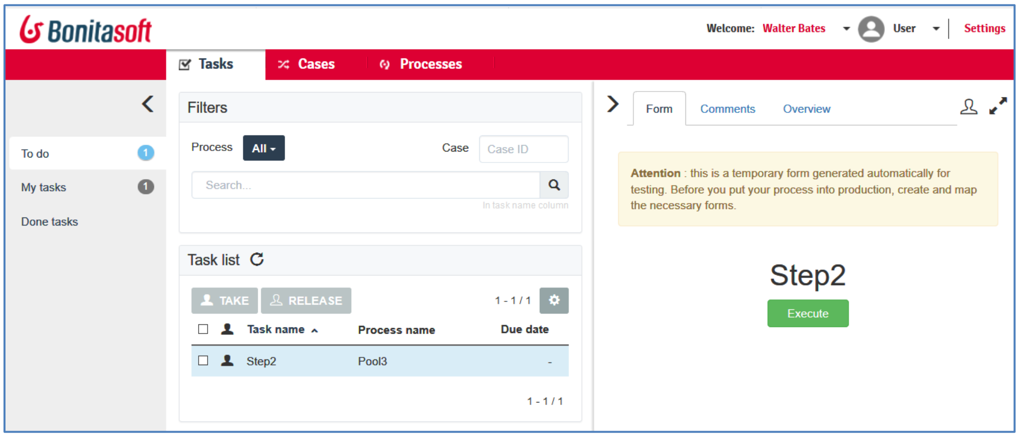


Attention : this is a temporary form generated automatically for testing. Before you put your process into production, create and map the necessary forms.

Task name	Process name	Due date
Step1	Pool2	-

Figure 29.   The human task Step 1 is listed on Bonita BPM portal

After we executed the first process, the second process is run and the task list is updated. Figure 30 shows the second business process involves the Step 2 activity, ready to be executed. This demonstration shows us how the messaging is used in Bonita BPM to start another business process.



Attention : this is a temporary form generated automatically for testing. Before you put your process into production, create and map the necessary forms.

Task name	Process name	Due date
Step2	Pool3	-

Figure 30.   The human task Step 1 is listed on Bonita BPM portal

THIS PAGE INTENTIONALLY LEFT BLANK

IV. COMPARISON AND RESULTS

This chapter identifies the differences in development effort between the two applications and discusses the limitations of Bonita BPM based on the applications we developed for this study. Its primary finding is that the persistence feature in Bonita BPM facilitates the application development because it automates the manual coding steps that we did in the persistence application in Java EE. As a result, this feature reduced the amount of programming effort required to develop the application. Likewise, Bonita BPM messaging feature allowed us to readily use BPMN messaging elements to send events messages to other business process. We did not need to write code for messaging in Bonita BPM as we had to do in Java EE messaging application. However, the capabilities of the BPM come with a price: namely, they limit developers' ability to use technologies not integrated into the BPMS.

A. JAVA EE AND BONITA BPM PERSISTENCE APPLICATIONS ANALYSIS

We found that developing the persistence application using Bonita BPM required less programming effort, which in turn reduced the time required for development comparing to use Java EE. In the Java EE persistence application, we used Hibernate as an ORM provider to map the Thesis class properties to the columns in the relational data table. We therefore performed multiple steps to develop the application: we had to manually write the SQL code to create the database table, we had to write XML code to configure multiple XML files, and we had to write the Java classes required for persistence.

By contrast, Bonita BPM, we could create and define business objects with their properties in BDM, and the Bonita BPM engine automatically generated the Java classes from the business objects, as well as the data table required to persist data. Bonita BPM also mapped class properties by using Hibernate with their proper column in the relational data table. We therefore did not create any code related to persistence in the Bonita BPM persistence application. There was likewise no need to create any XML configuration file as we had to do in Java EE—an example of how Bonita BPM accelerates application development.

This experiment revealed that a key difference between the two applications is Bonita BPM's automation of the manual programming effort required by a developer to set up the mapping using Hibernate via BDM capability. However, this streamlining of the mapping process comes at a cost: namely, that Bonita BPM limits developers to its integrated ORM technology, Hibernate. On the software development level, there are many ORM technologies other than Hibernate that developers can use for mapping, including Toplink, Speedment, or MyBatis. In this study, when we developed the Bonita BPM persistence application, we used the BDM capabilities to map the business objects to a relational table in the H2 database; we saw that mapping is automated in Bonita BPM because Hibernate is configured within Bonita BPM. However, what if we prefer to use an ORM tool other than Hibernate?

This question may lead us to examine the flexibility that Bonita BPM—or BPMSs in general--can offer to use a different ORM provider. Java EE is a flexible development platform that enables developers to use any compatible ORM tool: Speedment, for example, uses Java code for persistence (Speedment, 2018). According to Speedment, Speedment ORM provides faster performance for data persistence. Because of Java EE's flexibility, developers are enabled to use and test newer ORM providers like Speedment ORM, which is not the case when it comes to Bonita BPM. If a BPMS like Bonita BPM does not effectively support the use of an ORM tool like Speedment rather than Hibernate, . then it limits developers' ability, which in turn could be a reason for them to not adopt BPMSs to develop applications.

B. JAVA EE AND BONITA BPM MESSAGING APPLICATIONS ANALYSIS

We found that developing the messaging application using Bonita BPM required less effort compared to Java EE. In the Java EE messaging application, we used JMS API to send a message to a queue through a managed bean, which is then received by a message-driven bean. We performed multiple steps to develop the application, including manually configuring the Wildfly application server, creating the queue in ActiveMQ, writing the

Java code for the Java managed bean and message-driven bean. We also wrote the JSF code for web page.

By comparison, using Bonita BPM took less time to develop a simple messaging application. We created two business processes by dragging and dropping BMPN elements, and we used throw and catch messages elements to send and receive the message. The message is a start event, so, when the first business process sends the message, the second business process starts when its receives the message. We therefore did not need to create any code related to the messaging application in Bonita BPM. There was likewise no need to configure a messaging provider like ActiveMQ.

However, as with the persistence applications, Bonita’s simplicity comes at a cost: while it has the ability to integrate various messaging applications, it cannot be used as a tool to create messaging applications other than those messaging technologies for which it is already configured. Some enterprise applications require a high performance messaging provider that handles large amounts of data. Apache Kafka is an example of a messaging provider used to process and manage large streams of messages. Kafka could be used in GPS applications to exchange the location data in real time. However, Java EE or any programming platform is suitable to developing and integrating enterprise applications that require specific technology for messaging like Apache Kafka because it is a multi-purpose application development platform. Bonita BPM provides only the integration feature with other applications that use a specific messaging technology.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION AND FUTURE STUDY

This study revealed the differences between enterprise application development using BPMS versus a conventional programming platform. We used Bonita BPM as an example of BPMS and Java EE as an example of a traditional application development platform, creating a persistence application and a messaging application using each platform. This study revealed that BPMS reduced the amount of programming effort required to develop the applications. However, there are tradeoffs: BPMS limits developers to preconfigured technologies, whereas Java is more flexible, allowing developers to use any compatible technology but requiring more code.

That said, it remains to be investigated whether the same relative levels of effort hold true in the case of other types of important enterprise applications.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A

SQL commands to create Thesis table.

```
CREATE DATABASE IF NOT EXISTS samplejsfhibernate;

/* SQL Command To Use The Database */
USE samplejsfhibernate;

/* DROP Any Exisiting Table In The Database With Name As
"THESIS" */
DROP TABLE IF EXISTS THESIS;

/* SQL Command To Create The Table In A Database */

CREATE TABLE THESIS (
     ID INTEGER GENERATED ALWAYS AS IDENTITY NOT NULL,
     NAME VARCHAR(25),
     CURRICULUM INTEGER,
     ADVISOR VARCHAR(25),
     SECOND_READER VARCHAR(25),
     ACADEMIC_ASSOCIATE VARCHAR(25),
     GRADUATION_DATE DATE,
     PROBLEM_STATEMENT VARCHAR(25),
     RESEARCH_HYPOTHESIS VARCHAR(25),
     PRIMARY KEY (ID)
);
```

Thesis Java class

```
package main.java.com.jsfhibernate.pojo;
import java.util.List;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.bean.SessionScoped;
import javax.persistence.Column;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;


import
main.java.com.jsfhibernate.dao.DatabaseOperations;
import java.util.Date;
@ManagedBean
@RequestScoped
public class Thesis implements java.io.Serializable {

     private int id;
     private String name;
     private int curriculum;
     private String advisor;
     private String secondReader;
     private String academicAssociate;
     private Date graduationDate;
     private String problemStatement;
     private String researcHypothesis;
     public static DatabaseOperations dbObj;
     private static final long serialVersionUID = 1L;
     public int getId() {
          return id;
     }
     public void setId(int id) {
          this.id = id;
     }
     public String getName() {
          return name;
     }
     public void setName(String name) {
          this.name = name;
     }
     public int getCurriculum() {
          return curriculum;
     }
```

```
public void setCurriculum(int curriculum) {
     this.curriculum = curriculum;
}
public String getAdvisor()
{
     return advisor;
}
public void setAdvisor(String advisor)
{
     this.advisor = advisor;
}
public String getSecondReader()
{
     return secondReader;
}
public void setSecondReader(String secondReader)
{
     this.secondReader = secondReader;
}
public String getAcademicAssociate()
{
     return academicAssociate;
}
public void setAcademicAssociate(String
academicAssociate)
{
     this.academicAssociate = academicAssociate;
}
public Date getGraduationDate()
{
     return graduationDate;
}
public void setGraduationDate(Date
graduationDate)
{
     this.graduationDate = graduationDate;
}
public String getProblemStatement()
{
     return problemStatement;
}
public void setProblemStatement(String
problemStatement)
{
     this.problemStatement = problemStatement;
}
```

```
          public String getResearcHypothesis()
          {
               return researcHypothesis;
          }

          public void setResearcHypothesis(String
          researcHypothesis)
          {
               this.researcHypothesis = researcHypothesis;
          }

          // Method To Add New Student Details In Database
          public void saveThesisRecord() {
               System.out.println("Calling
               saveThesisRecord() Method To Save Student
               Record");
               dbObj = new DatabaseOperations();
               dbObj.addThesisInDb(this);

          }
     }
```

Hibernate configuration file.

```
     <hibernate-configuration>
        <session-factory>
        <property name="connection.driver_class">
         org.apache.derby.jdbc.ClientDriver </property>
     <property
name="connection.url">jdbc:derby://localhost:1527/
samplejsfhibernate;create=true;</property>
     <property
name="hibernate.connection.username">test</property>
     <property
name="hibernate.connection.password">test</property>
     <property
name="dialect">org.hibernate.dialect.DerbyDialect</property
>
     <property
name="hibernate.generate_statistics">true</property>
     <property name="show_sql">true</property>

     <property
name="hibernate.current_session_context_class">org.hibernat
e.context.internal.ThreadLocalSessionContext</property>

<mapping resource="Thesis.hbm.xml"/> </session-factory>

</hibernate-configuration>
```

Hibernate configuration file.

```
<hibernate-mapping>
    <class name="com.jsfhibernate.pojo.Thesis"
    table="THESIS">

<id name="id" type="java.lang.Integer">
<column name="id" />
<generator class="identity" />
</id>

<property name="name" type="string">
<column name="Name" />
</property>
<property name="curriculum"   type="java.lang.Integer">
<column name="Curriculum" />
</property>

<property name="advisor" type="string">
<column name="Advisor" />
</property>

<property name="secondReader" type="string">
<column name="Second_Reader" />
</property>

<property name="academicAssociate" type="string">
<column name="Academic_Associate" />
</property>

<property name="graduationDate" type="java.util.Date">
<column name="Graduation_Date" />
</property>


<property name="problemStatement" type="string">
<column name="Problem_Statement" />
</property>

<property name="researcHypothesis" type="string">
<column name="Research_Hypothesis" />
</property>

</class>
```

```
</hibernate-mapping>
```

Hibernate Utility class

```
package main.java.com.jsfhibernate.util;
```

```
     import org.hibernate.SessionFactory;
     import org.hibernate.cfg.Configuration;
     public class HibernateUtil {
     private static SessionFactory sessionFactoryObj =
buildSessionFactoryObj();
// Create The SessionFactory Object From Standard
//(Hibernate.cfg.xml) Configuration File
public static SessionFactory buildSessionFactoryObj() {
try {
     sessionFactoryObj = new
Configuration().configure().buildSessionFactory();
               }
catch (ExceptionInInitializerError exceptionObj) {
     exceptionObj.printStackTrace();
               }
     return sessionFactoryObj;
          }
public static SessionFactory getSessionFactory() {
               return sessionFactoryObj;
          }
}
```

Database Operations class

```
     package main.java.com.jsfhibernate.dao;
     import java.util.ArrayList;
     import java.util.List;
     import javax.faces.context.FacesContext;
     import org.hibernate.Query;
     import org.hibernate.Session;
     import org.hibernate.Transaction;
     import main.java.com.jsfhibernate.pojo.Thesis;
     import main.java.com.jsfhibernate.util.HibernateUtil;
     public class   DatabaseOperations {
          private static Transaction transObj;
          private static Session sessionObj =
HibernateUtil.getSessionFactory().openSession();

          // Method To Add New Thesis Details In Database

          public void addThesisInDb(Thesis thesisObj) {

               try {

                    transObj =
                    sessionObj.beginTransaction();
                    sessionObj.save(thesisObj);
                    System.out.println("Thesis Record With
     Id: " + thesisObj.getId() + " Is Successfully Created
     In Database");

     // XHTML Response Text
     //FacesContext.getCurrentInstance().getExternalContext
     ().getSessionMap().put("createdThesisId",
     thesisObj.getId());
     FacesContext.getCurrentInstance().getExternalContext()
     .getRequestMap().put("createdThesisId",
     thesisObj.getId());
     }
     catch (Exception exceptionObj) {
     exceptionObj.printStackTrace();
     }
     finally { transObj.commit();}
     }
```

User interface code in JSF

```
<html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html">
     <h:head>
     <meta charset="utf-8" name="viewport"
content="width=device-width, initial-scale=1" http-
equiv="X-UA-Conpatible" />
     <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/
bootstrap.min.css" />
         <title>Thesis Proposal</title>
         <style type="text/css">
          .btnPos {
               margin: 100px;
          }
          .marginLeft12 {
               margin-left: 100px;
          }
          .successText {
               padding-left: 100px;
               font-weight: bold;
               color: green;
          }
          .stud-table {
               border-collapse: collapse;
                      }
          .stud-table-header {
               text-align: center;
               background: none repeat scroll#E5E5E5;
               border-bottom: 1px solid #BBBBBB;
               padding: 2px;
               }
          .stud-table-row {
               padding: 100px;
               text-align: center;
               }
         </style>
     </h:head>
     <h:body>
         <center><h2>Create Thesis Proposal</h2></center>
     <h:form id="thesisSaveForm" styleClass="form-
horizontal">
     <h:panelGrid columns="2">
<div class="form-group">
```

```
<h:outputLabel value="Name:" styleClass="control-label" />
<div class="col-xs-20">
<h:inputText value="#{thesis.name}" id="thesisName"
styleClass="form-control" required="true"
requiredMessage="Please enter name." >
<f:attribute name="message" value="Please write something"
/>
</h:inputText>
</div>
<h:message for="thesisName" style="color:red" /> </div>
<div class="form-group">
<h:outputLabel value="Curriculum:" styleClass="control-
label" />
<div class="col-xs-20">
                         <h:selectOneMenu value =
"#{thesis.curriculum}" styleClass="form-control" >
                                 <f:selectItem itemValue =
"1" itemLabel = "1" />
<f:selectItem itemValue = "2" itemLabel = "2" />
<f:selectItem itemValue = "3" itemLabel = "3" />
<f:selectItem itemValue = "4" itemLabel = "4" />
<f:selectItem itemValue = "5" itemLabel = "5" />
<f:selectItem itemValue = "6" itemLabel = "6" />
<f:selectItem itemValue = "7" itemLabel = "7" />
<f:selectItem itemValue = "8" itemLabel = "8" />
<f:selectItem itemValue = "9" itemLabel = "9" />
<f:selectItem itemValue = "10" itemLabel = "10" />
</h:selectOneMenu>
</div> </div>

<div class="form-group">

<h:outputLabel value="Advisor:" styleClass="control-label"
/>

<div class="col-xs-20">

<h:inputText value="#{thesis.advisor}" styleClass="form-
control" /></div> </div>

<div class="form-group">

<h:outputLabel value="SecondReader:" styleClass="control-
label" />

<div class="col-xs-20">

<h:inputText value="#{thesis.secondReader}"
styleClass="form-control" /> </div>

                 </div>
```

```
     <div class="form-group">
     <h:outputLabel value="AcademicAssociate:"
styleClass="control-label" />
<div class="col-xs-20">
<h:inputText value="#{thesis.academicAssociate}"
styleClass="form-control" />
</div></div>
<div class="form-group">
<h:outputLabel value="Date of Graduation:"
styleClass="control-label" converterMessage="Format must
be: yyyy-MM-dd" /
<div class="col-xs-20">
<h:inputText id="graduationDate"
value="#{thesis.graduationDate}" styleClass="form-control"
>
<f:convertDateTime pattern="yyyy-MM-dd"/></h:inputText>
</div>
<h:message for="graduationDate" style="color:red" />
</div>
<div class="form-group">
<h:outputLabel value="Problem Statement:"
styleClass="control-label" /> <div class="col-xs-20">
<h:inputText value="#{thesis.problemStatement}"
styleClass="form-control" /></div>
</div>
<div class="form-group">
<h:outputLabel value="Research Hypothesis:"
styleClass="control-label" />
                    <div class="col-xs-20">
                         <h:inputText
value="#{thesis.researcHypothesis}" styleClass="form-
control" />
     </div>
 </div>
<div class="form-group">
<div class="col-xs-offset-2 col-xs-20">
```

```
<h:commandButton value="Save Record"
action="#{thesis.saveThesisRecord}" styleClass="btn btn-
primary btn-sm btnPos" />

</div></div>

</h:panelGrid>

<h:outputText id="saveResult" rendered="#{not empty
createdThesisId}" value="!! Thesis Record Saved In Database
!!" styleClass="successText" />

</h:form><br /><br />

</h:body>

</html>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B

Java Managed Bean. Adapted from (JBoss, n.d.)

```
import java.util.Properties;
import javax.jms.Connection;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.MessageProducer;
import javax.jms.Queue;
import javax.jms.QueueConnectionFactory;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.QueueConnection;
import javax.jms.QueueSession;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.annotation.Resource;

public class Request {
 @Resource(lookup ="java:jboss/exported/jms/
RemoteConnectionFactory")
 ConnectionFactory connectionFactory;
 @Resource(lookup = "java:/jms/queue/HelloQueue")
 Destination destination;

private String msg;

public String sendMessage() {
System.out.println("ConnectionFactory"+connectionFactory);
System.out.println("Destination"+destination);
Properties output;
try {
QueueConnection queueConnection = (QueueConnection)
connectionFactory.createConnection("***","**");
System.out.println("QueueConnection"+queueConnection);
QueueSession queueSession =
queueConnection.createQueueSession(false,Session.AUTO_ACKNO
WLEDGE);
System.out.println("QueueSession"+queueSession);
MessageProducer messageProducer =
queueSession.createProducer(destination);
System.out.println("MessageProducer"+messageProducer);
TextMessage textMessage =
queueSession.createTextMessage("Text");


System.out.println("TextMessage"+textMessage);
messageProducer.send(textMessage);
System.out.println("Message is sent!"+ msg);
}
catch (Exception ex) {
System.out.println("Connection exception"+ex);
}

return "success";
}
public String getMsg() {return msg;}
     public void setMsg(String msg) {
          this.msg = msg;
     }
}
```

Java message-driven bean. Adapted from (JBoss, n.d.)

```
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.Message;
import javax.jms.MessageListener;
import java.util.logging.Logger;
import java.util.logging.Level;

@MessageDriven(activationConfig = {
@ActivationConfigProperty(propertyName = "destination",
propertyValue = "jms/queue/HelloQueue"),
@ActivationConfigProperty(propertyName = "destinationType",
propertyValue = "javax.jms.Queue")
},
mappedName = "jms/queue/HelloQueue")
public class SimpleMessageBean implements MessageListener {
static final Logger logger =
Logger.getLogger("SimpleMessageBean");
    public SimpleMessageBean() {

    }

    public void onMessage(Message message) {
     System.out.println("Message is received: start process
");
    }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Allègre, R. (2013). *Develop a first process application* . Retrieved November 20, 2018, from BonitaSoft: https://www.bonitasoft.com/landing/down/EN/ Develop_a_First_Process_Application_EN.pdf

Apache. (2018). *Apache Tomcat*. Retrieved November 10, 2018, from http://tomcat.apache.org/

Appian. (2018). *low-code guide*. Retrieved November 20, 2018, from Appian: https://www.appian.com/resources/low-code-guide/

Bonitasoft. (2009). *Bonitasoft*. Retrieved November 18, 2018, from https://www.bonitasoft.com/bonita-platform

Bonitasoft. (n.d.a). *Boniasoft Documentation: BDM Management in Bonita Portal*. Retrieved 11 18, 2018, from Bonitasoft: https://documentation.bonitasoft.com/ bonita/7.7/bdm-management-in-bonita-bpm-portal

Bonitasoft. (n.d.b). *Bonitasoft Documentation - Connectors*. Retrieved November 10, 2018, from Bonitasoft: https://documentation.bonitasoft.com/bonita/7.7/ connectors-overview

Bonitasoft. (n.d.c). *Bonitasoft Documentation : Documents*. Retrieved November 15, 2018, from Bonitsoft: https://documentation.bonitasoft.com/bonita/7.7/documents

Bonitasoft. (n.d.d). *Bonitasoft Documentation: Reporting overview*. Retrieved November 10, 2018, from Bonitasoft: https://documentation.bonitasoft.com/bonita/7.7/ reporting-overview

Bonitasoft. (n.d.e). *Bonitasot Documentation: Tasks*. Retrieved November 10, 2018, from Bonitasoft: https://documentation.bonitasoft.com/bonita/7.7/diagram-tasks

Bonitasoft. (n.d.f). *Expense Report Example*. Retrieved November 10, 2018, from Bonitasoft: https://community.bonitasoft.com/project/expense-report-example1

Capgemini . (2014). *A Productivity Comparison of Pegasystems Pega 7 versus Java Enterprise Edition Custom Build*. Retrieved November 20, 2018, from Pega: https://www1.pega.com/insights/resources/productivity-comparison-pegasystems-pega-7-versus-java-enterprise-edition-custom

Chang, J. F. (2006). *Business Process Management Systems Strategy and Implementation.* Boca Raton, FL: Auerbach Publication.

Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Devika Gollapudi, Kim Haase, William Markito, Chinmayee Srivathsa. (2013a). *The Java EE 6 Tutorial - Basic JMS API Concepts*. Retrieved from Oracle: https://docs.oracle.com/javaee/6/tutorial/doc/bncdx.html

Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Devika Gollapudi, Kim Haase, William Markito, Chinmayee Srivathsa. (2013b). *The Java EE 6 Tutorial - Chapter 32- Introduction to the Java Persistence API*. Retrieved November 10, 2018, from Oracle: https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html

Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Devika Gollapudi, Kim Haase, William Markito, Chinmayee Srivathsa. (2013c). *The Java EE 6 Tutorial - Differences between Java EE and Java SE*. Retrieved November 1, 2018, from Oracle: https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html#gcrkk

Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Devika Gollapudi, Kim Haase, William Markito, Chinmayee Srivathsa. (2013d). *The Java EE 6 Tutorial - Java EE 6 APIs*. Retrieved November 20, 2018, from Oracle: https://docs.oracle.com/javaee/6/tutorial/doc/bnacj.html

Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Devika Gollapudi, Kim Haase, William Markito, Chinmayee Srivathsa. (2013e). *The Java EE 6 Turtorial - The JMS API Programming Model*. Retrieved Octomber 10, 2018, from Oracle: https://docs.oracle.com/javaee/6/tutorial/doc/bncdx.html

Ferreira, D. R. (2013). *Enterpries Systems Integration A Process-Oriented Approach.* New York, NY: Springer.

Getnet Amene Yeshanew, Mapinduzi Muhochi. (2010). *Master Thesis: Opportunities and Challenges of Business Process Management Systems (BPMS).* Lund University.

Harmon, P. (2014). *Business Process Change*. Third Edition. Waltham, MA:Morgan Kaufmann.

JBoss. (n.d.). Retrieved November 20,2018 from JBoss: http://docs.jboss.org/jbossmessaging/docs/guide/html/examples.html

John Footen, J. F. (2008). *The Service-Oriented Media Enterprise: SOA, BPM, and Web Services in Professional Media Systems.* Burlington, MA: Elsevier.

Mike Kieth, Merrick Schincariol, Massimo Nardone. (2018). *Pro JPA 2 in Java EE 8 - An In Depth Guide to Java Persistence APIs*. New York, NY: Apress.

Netbeans. (2018). Netbeans. Retrieved November 18, 2018, from www.netbeans.org

OMG. (2011). Business Process Model and Notation. Retrieved November 15, 2018, from Open Management Group: https://www.omg.org/spec/BPMN/2.0/

O'Neil, E. J. (2008). Object/relational mapping 2008: hibernate and the entity data model (edm). *SIGMOD '08 Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 1351-1356). Vancover, Canada: ACM.

Oracle. (n.d.). *Oracle*. Retrieved November 25, 2018, from Oracle: https://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp-136984.html

Ozill. (2015). *Bonitasoft*. Retrieved November 20, 2018, from Youtube: https://www.youtube.com/watch?v=nS-aq-DbFrE

Redhat. (2017). *What is WildFly?* Retrieved October 18, 2018, from WildFly: wildfly.org/about/

Richards, Mark; Haefel, Richard Monson; Cappell, David A. (2009). *Java Message Service.* Sebastopol, California: O'reilly.

Speedment. (2018). Speedment. Retrieved November 15, 2018, from https://www.speedment.com/

Stancapiano, L. (2017). *Mastering Java EE Development with Wildfly.* Birmingham, UK: Packt Publishing.

Stiehl, V. (2013). *Process-Driven Appliations with BPMN.* Walldorf, Germany:Springer.

Sun, A. (2016). *Comparison of Java Persistence Layer Technologies.* Retrieved October 15, 2018, from https: http://www.diva-portal.se/smash/get/diva2:1040755/ FULLTEXT01.pdf

Wenqiong Cui , Yaohan Liu. (2010). *Master Thesis: Factors Affecting Business Process Management System Adoption and Diffusion.* Lund University, Lund, Sweden.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California