Project #3

Enterprise Application Integration

João Ribeiro
David Cruz
Filipe Norte

07/12/2012

1 Introduction

The current report provides a description of the solution implemented for project 3. It includes static and dynamic perspectives of the architecture and an explanation of some of the major design decisions.

2 Architecture Description

Dynamic View

The figures below show the dynamic views of the system.

Service working flow

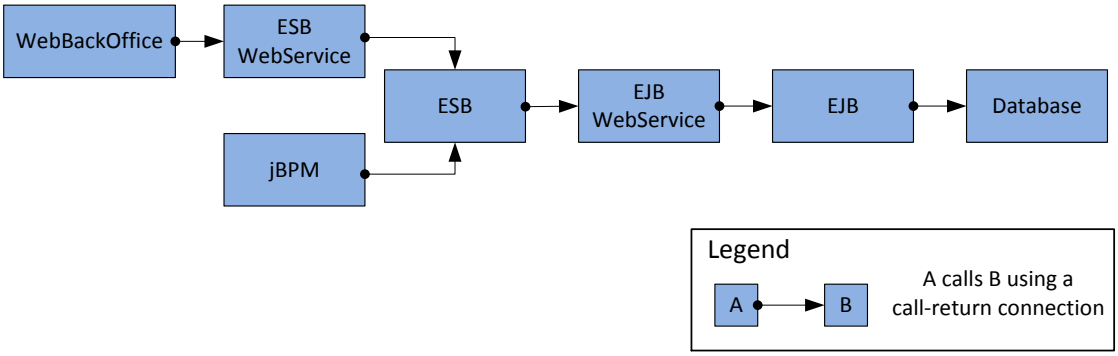

Figure 1 – Dynamic view of the system

In the WebBackOffice we simulate a user requesting information that is only accessible using an ESB service. We use web services to allow the web site to communicate with the ESB services and use its functionalities. Moreover, we created a new EJB to provide a web service interface over some of the Project#2 system (getUsersFollowingCompany(), `getUserEmailCount()` and incrementUserEmailCountFromList()) to make its functionalities available for external entities like the new ESB services used by the jBPM.

Business Process Management

The BPM process diagram is represented in Figure 2. It shall be noted that in this diagram some nodes appear identified as “node” but they are actually calling ESB services. This is true for *Send e-Mail*, *Count e-Mail* and *SubmitReport* nodes. The orchestration service enters the *start* state with a message carrying company stock information, which is then processed by the process.

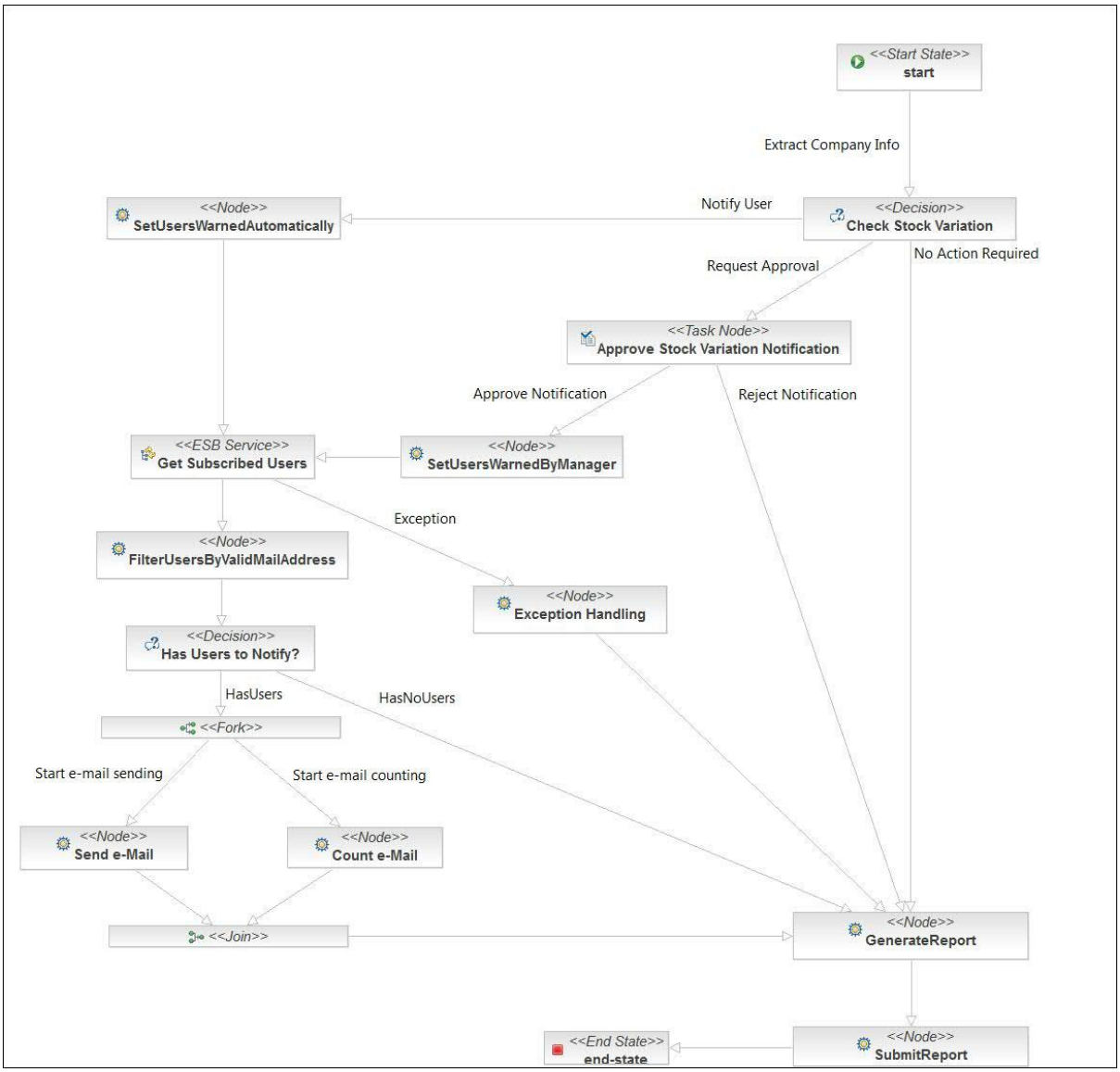

Figure 2 – Dynamic view of the Business Process Management process

Static View

The project is composed by different packages with different responsibilities on the overall project. The following table gives a brief overview of their responsibilities and relevant information:

Project	Package	Responsibilities
EAICommon	**eai.msejdf. config**	Handles all the configuration settings required by the system
	eai.msejdf. data	The data model object that was used internally by the system. Contains the data model xsd schema definition.
	eai.msejdf. esb	Package responsible for handling the data models used on the ESB services

	eai.msejdf. utils	Package responsible for various utilities used by the system. Includes classes for handling the xml/xsl transformations, property loading, string operations and sending emails.
	eai.msejdf. web	Responsible for handling the session management
EJBServer	**eai.msejdf.admin**	Package to handle the BackOffice admin operations
	eai.msejdf.jms	This package is responsible for handling the Java Messaging System (JMS) interactions (receiving of messages through JMS)
	eai.msejdf.security	Package for security related calls, such as login, registration, etc.
	eai.msejdf.timer	Package to handle the operations that are timely scheduled
	eai.msejdf.user	Package to handle the FrontOffice user operations
	eai.msejdf.webservices	Package to handle the web services provided by the EJBs
EJBServerInterfaces	**eai.msejdf.admin**	Provides the bean interfaces for the admin operations
	eai.msejdf.exception	Implementation of specific exceptions
	eai.msejdf.security	Provides the bean interfaces for security related operations
	eai.msejdf.sort	The Enum that defines the order by clauses.
	eai.msejdf.user	Provides the bean interfaces for the user operations
	eai.msejdf.webservices	Provides the bean interfaces for the web services operations
ESBServices	**eai.msejdf.esb**	Provides ESB functionalities
ESBWebServices	**eai.msejdf.esb.webservice**	Provides a web service interface for the ESB services
JBPMServices	**jpdl**	Defines the orchestration flow
	eai.msejdf.jbpm.actions	Define the functionality support for the orchestration action
JPAData	**eai.msejdf.persistence**	JPA Entity implementations
WebBackOffice	**eai.msejdf.web.backoffice.admin**	Admin related integrations with Logic Tier
	eai.msejdf.web.backoffice.security	Security related interactions with Logic Tier
	Webcontent	Web site definition
WebFrontOffice	**eai.msejdf.pagedata**	Core classes per data/operation type for interactions with Logic Tier. See design decisions
	eai.msejdf.validator	Data validation operations
	eai.msejdf.pages	Beans (non EJB) for each HTML page containing code for interactions with the EJBs. Derived from pagedata classes. See design decisions
	Webcontent	Web site definition

Table 1 – Package list and responsibilities

Some of these packages are composed of different structures and classes that define object instances used internally by the systems. A brief detail of some design decisions associated with them will be described in the following sections for the main or most structured packages.

Design Decisions

The following sections list some major design decisions.

BPM Orchestration

- To support the calls to the ESB services and implementation of the required functionality, the parameters to be supplied to the services, as well as the return values, need to be adapted/transformed between calls to different ESB services as each service's interface has its own signature. These conversions were done in the BPM context (example: extract company name or variation percentage from company information in the message). Consequently, the orchestration process service contains some classes to support such transformations or context setup, as the nodes in Figure 2 *SetUsersWarnedAutomatically* and *SetUsersWarnedByManager*, and the transition *Extract Company Info* are representative examples of. Although some are essential for the process flow control, the services could have been designed to avoid such transformations in most cases. However, the consequences would be that they would be tied to this orchestration specifically, minimizing the possibility of their reuse by another orchestration process. For this reason, such extra classes are part of the orchestration so the ESB services could be designed with a more generic interface. In addition, specific ESB services were not designed for such transformations, as it would increase the processing overhead associated to the ESB service calls (invocation, extra serialization and deserialization of data, etc.) and would lead to extra support services that do not really provide functionality that justifies the existence of a service, polluting the architecture with minor "support" services (affects maintainability).
- Some of the extra processing classes are modeled explicitly in the diagram of Figure 2 through nodes (*SetUsersWarnedAutomatically* and *SetUsersWarnedByManager*) while other are implicitly through transitions (*Extract Company*). Explicit support nodes could have been also modeled by transitions to make the flow cleaner from a visual/logical flow perspective. However, to make the process easier to follow regarding the intended flow and required actions, we decided to make them explicit for the purpose of this assignment, using the implicit representation in the transitions for minor support actions less "relevant" from the business flow perspective.
- We decided to implement exception handling in the orchestration process only for one ESB service call (*Get Subscribed Users* node). Although necessary in a real case scenario, considering our business constraint (time) and the learning purpose of this assignment, implementing the exception handling process at least once would already allow us to assimilate its handling when interfacing the ESB. Adding it to other nodes invoking ESB services would be done in an identical way. It shall however be noted that the complexity would be slightly higher for the *Send e-mail* and *Count e-mail* nodes due the parallel execution and associated failure combinations, which would have to result in rollback (if possible) or other corrective actions.

Splitter and aggregator

For the current project we decided to implement our own splitter, because we saw that we were unable to split the companies in different messages. The following points indicate the split process:

- The class eai.msejdf.esb.SplitCompaniesAction, receives a list of companies through the message bus and prepares the AggregationDetails to send to the aggregator for each message to be send. The aggregator tag follows the structure defined in these situations ***<UUID>:<number inside the series>:<total messages in the series>:<timestamp>:<origin class>***
- The message is then sent to the jBPM orchestration process that processes each company and produces an individual report for each company.
- The message with the individual report is then received by the aggregator that joins all the messages in one global one with all the chunks stored through the attachments
- These attachments are then aggregated into one object message by the class eai.msejdf.esb.AggregatedAssemblerAction that produces a list of individual reports to be calculated globally.
- The class eai.msejdf.esb.ReportAggregateAction, is responsible for reading all the individual reports and produce a final message to the report listener for printing. The message is sent using the already existent ESB class for sending JMS messages.

3 Participations

The participation by each team member is listed as the packages they mainly worked on. However, each one also worked on other packages, although to less extend in order to update with functionalities or for corrections.

João Ribeiro (Time 81 hours, 34 minutes)

- Webservices ESB, Webservices EJB, WebBackOffice

David Cruz (91 hours, 43 minutes)

- Spliter, Aggreagator, Webservices ESB, Webservices EJB, ReportListener, Webprobe Replicator

Filipe Norte (Time 94 hours, 55 minutes)

- JBPMServices (Orchestrator), ESBServices

Annex A – Project deployment instructions

Prerequisites

In order to execute this software it is necessary to have JMS with JBoss AS 7 installed, MySql Server version 5 running and JBoss ESB server 4.10 with the configurations accordingly. The custom settings from the previous project should be set according to the previous report. Only the new ones will be described in this document, on the next section.

Deploy directory structure

The deploy directory structure is set in the following manner:

Dir	Meaning
Ear	Contains the EAR package to deploy to the JBoss 7 instance. Deploy all the components necessary to work with project (EJB, Webservice, Web Frontend and Backend).
Esb	Contains the ESB package to deploy to the JBoss ESB 4.10 server instance. Deploy all the components necessary to work with project (ESB and ESB webservice).
Jbpm	Contains the Jbpm process archive package to deploy to the JBoss ESB 4.10 server instance. Deploy all the components used on the Jbpm process.
Src	All the source code of whole the application

Table 2 – Deployment directory structure

Jbpm configuration

The configuration of the jBPM orchestration process consists simply in its deployment in the JBoss ESB server, which is done by uploading the process file present in the *Jbpm* folder previously reference through the site http://localhost:8180/jbpm-console/app/processes.jsf . Once completed, no further action is required.

Settings and Configurations

The following section describes the settings stored in configuration files, their meaning and default values (the previous settings from the P1 and P2 project will not be handled in this report).

config.properties

This file contains the main application settings properties of the system. The following table describes the main keys of this file.

Setting	Default value	Meaning
app.esbreplicator.jndi.provider	jnp://127.0.0.1:1199	The jndi address for the JBoss 4.10 ESB server
app.esbreplicator.topic	topic/EAIProj3_gateway	The JBoss ESB topic name for replication of the message probe
app.esbreporter.queue	queue/report_queue	The JBoss ESB queue for listening the reports produced by the process

Table 3 – Configuration settings