FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Using Sun Java Composite Application Platform Suite (Java CAPS) for Enterprise Application Integration

Ricardo Alves da Silva

Report of Project
Master in Informatics and Computing Engineering

Supervisor: Gil Manuel Magalhães de Andrade Gonçalves (Engineer)

14th July, 2009

Using Sun Java Composite Application Platform Suite (Java CAPS) for Enterprise Application Integration

Ricardo Alves da Silva

Report of Project
Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: José Manuel de Magalhães Cruz (Doctor)

External Examiner: Feliz Ribeiro Gouveia (Doctor)

Internal Examiner: Gil Manuel Magalhães de Andrade Gonçalves (Engineer)

14th July, 2009

Abstract

Presently, given the high competitiveness felt in the business world, organizations are increasingly investing in Information Technologies (IT) for knowledge sharing through the storage, processing, production and communication of data. The problem is that typically this information is not centralized in a single system, having the organizations a wide variety of applications and systems for performing different tasks. Although belonging to the same company, these systems are most of the times disparate and not compliant with each other. It is to address this problem that emerges the concept of Enterprise Application Integration (EAI), which consists in the process of connection between different business applications, to promote a fast and reliable exchange of business processes and data. The retail world, where this project is inserted, is also an example of this scenario, given the constant changes and updates made to the retailers' Enterprise Resource Planning (ERP) applications, aiming to optimize their business model.

The work described in this report has been developed for a client who belongs to the largest international consortium of retailers concerning the number of stores spread throughout the world. It is a project in the EAI area, characterized by the implementation of data integration flows between the Oracle Retail Merchandising System (ORMS) and a proprietary ERP of the retailer, the IMAge. Since the IMAge represents a pilot-project, a gradual migration between these systems is desirable, and a recommended integration based in the Publication/Subscription model, using Java Composite Application Platform Suite (Java CAPS), from Sun Microsystems, as main technology. This tool provides a standards-based, open, extensible platform for developing software infrastructures using a Service-Oriented Architecture (SOA) approach.

This project has been developed at Wipro Retail, one of the largest companies for consulting and implementation of information systems in the area of retail. During the period length of the project was possible to deal with some of the latest distribution and integration technologies used in the market of retailing, which are described throughout the report. The study of the technologies involved in the project, the implemented solutions implemented and the problems faced, all this combined with the integration of the EAI team constituted for the client, proved to be a very profitable and enriching experience, not only in the functional scope of retail, but also at technological level.

Resumo

Actualmente, tendo em conta a elevada competitividade sentida no mundo dos negócios, as organizações estão a investir cada vez mais em Tecnologias de Informação (TI) para partilha de conhecimento através do armazenamento, processamento, produção e comunicação de dados. O problema é que tipicamente esta informação não se encontra centralizada num único sistema, possuindo as organizações uma ampla variedade de aplicações e sistemas para a realização de diferentes tarefas. Ainda que pertencendo à mesma empresa, estes sistemas são muitas vezes dispares e em não conformidade entre eles. É no sentido de dar resposta a este problema que surge a Integração de Aplicações Empresariais (EAI), a qual consiste no processo de ligação entre diferentes aplicações empresariais, no sentido de promover uma troca fiável e rápida de processos de negócio e dados. O mundo do retalho, onde o presente projecto está inserido, é também um exemplo deste cenário, tendo em conta as constantes alterações e actualizações efectuadas às aplicações de Enterprise Resource Planning (ERP) dos retalhistas, visando optimizar o seu modelo negócio.

O trabalho descrito neste documento foi desenvolvido para um cliente que pertence ao maior consórcio internacional de retalhistas no que diz respeito ao número de lojas que têm espalhadas pelo mundo. Trata-se portanto de um projecto na área de EAI, caracterizando-se pela implementação de fluxos de integração de dados entre o Oracle Retail Merchandising System (ORMS) e um ERP proprietário do retalhista, o IMAge. Uma vez que o IMAge representa um projecto-piloto é desejável uma migração gradual entre estes sistemas, sendo recomendada uma integração baseada no modelo de Publicação/Subscrição, usando como principal tecnologia o Java Composite Application Platform Suite (Java CAPS) da Sun Microsystems. Esta ferramenta disponibiliza uma plataforma aberta, extensível e baseada em padrões, para desenvolvimento de infra-estruturas de software, usando uma abordagem baseada numa arquitectura orientada a serviços (SOA).

Este projecto realizou-se numa das maiores empresas de consultoria e implementação de sistemas de informação na área do retalho, a Wipro Retail. Durante o período de duração do projecto foi possível lidar com algumas das mais recentes tecnologias de distribuição e integração usadas no mercado do retalho, as quais são descritas ao longo do relatório. O estudo das tecnologias envolvidas no projecto, as soluções implementadas e os problemas enfrentados, tudo isto aliado à integração da equipa de EAI constituída para o cliente, revelou-se uma experiência bastante proveitosa e enriquecedora, não só no âmbito funcional do retalho, mas também ao nível tecnológico.

Acknowledgments

Firstly the author wishes to thank his family, specially his parents, for every single effort and sacrifice made to ensure the best education for their son. Their life values together with their incessant love, affection, understanding and support, represent the greatest richness one can have, and words are surely not enough to express the author's gratitude to them.

Also, a very special acknowledgment to the Justiniano family, for the hosting provided during the period of the author's academic course. Certainly no money can pay their support, advice and friendship felt in all of the good times shared in the recent years.

To the institution of project, Wipro Retail, thanks for the opportunity and excellent conditions offered during the internship.

To Carla Almeida and Ana Raquel Lopes, for having followed the trainees during the recruitment process and their integration in the company, always available and guaranteeing that all necessary work conditions were gathered.

Thanks to Eng. Hugo Neto for his sympathy demonstrated throughout the recruitment interview and for having recognized potential in the author for performing his internship at Wipro Retail.

To everybody in the Wipro team gathered for this project's retailer, under the wisdom and comprehensive guidance of Dr. João Pinho, namely to my supervisor in the company, Eng. Rui Peixoto, a special acknowledgment for his support, availability and willingness continuously demonstrated throughout the internship, as well as for the other members of the integration team, Eng. André Guimarães, Eng. Marco Ferreira, Eng. Paulo Correia and Eng. Sérgio Xavier, for the ease of integration and sense of team spirit.

The author is also grateful to his FEUP supervisor, Eng. Gil Gonçalves, for his wise guidance and availability for the period of the project, as well as his dedication and suggestions on the writing of the present report.

For everyone at FEUP, namely Prof. Raul Vidal and Prof. Augusto Sousa, for their continuous efforts for bringing together the best for their students, as well as every teacher and colleague with whom the author had the privilege to learn and work during his academic training.

Finally, the author would like to extend his acknowledgments to all those who directly or indirectly gave their support and advice for the period of this project.

Contents

List of Figures

List of Tables

Abbreviations

ACC	Acceptance
API	Application Programming Interface
ASM	Application Support and Maintenance
BPEL	Business Process Execution Language
BPMN	Business Process Modeling Notation
CMF	Common Message Format
CMMI	Capability Maturity Model Integration
CRM	Customer Relationship Management
CRUD	Create, Replace, Update and Delete
CSV	Comma-Separated Values
DEV	Development
DBMS	Database Management System
DTD	Document Type Definition
EAI	Enterprise Application Integration
EDI	Electronic Data Interchange
ERP	Enterprise Resource Planning
FTP	File Transfer Protocol
IDE	Integrated Development Environment
IMAge	Integrated Management Application for Grocery Enterprises
IS	Information System
IT	Information Technology
JAR	Java Archive
JCAPS	Java Composite Application Platform Suite
JCD	Java Collaboration Definition
JMS	Java Message Service
MOM	Messaging Oriented Middleware
OLAP	Online Analytical Processing
OMG	Object Management Group
ORDF	Oracle Retail Demand Forecasting
ORDM	Oracle Retail Distribution Management
ORDW	Oracle Retail Data Warehouse
OReIM	Oracle Retail Invoice Matching
OReSA	Oracle Retail Sales Audit
ORMS	Oracle Retail Merchandising System
ORPAS	Oracle Retail Predictive Application Server
ORPM	Oracle Retail Price Management
ORSIM	Oracle Retail Store Inventory Management
ORTM	Oracle Retail Trade Management
ORWMS	Oracle Retail Warehouse Management System
OTD	Object Type Definition
POS	Point-of-Sale

RTV	Return-to-Vendor
SCM	Supply Chain Management
SOA	Service-Oriented Architecture
SQL	Structured Query Language
TAFR	Transformation, Address, Filtering and Routing
UAT	User Acceptance Test
UML	Unified Modeling Language
UTD	Unit Test Document
XML	Extensible Markup Language

1 Introduction

This first chapter introduces the project by presenting its scope, motivation and objectives, in order to identify and define the problems that the dissertation addresses. This section also describes briefly the project proposal, as well as the company where the work took place and its final customer. Finally, it is presented a brief summary of each of the subsequent chapters.

1.1 Scope

Given the current factors of competitiveness and market changes within the enterprises, today, more than ever, it is essential to have an investment in the area of the Information Technologies (IT). In fact, nowadays almost all business relationships between customers, suppliers and employees are based on Information Systems (IS).

Unfortunately, knowing the variety of interests of organizations, there isn't a system able to provide all the desired information, so, taking this into account, companies have different types of IS's according to their business needs. For this reason, such variety of applications is sometimes referred to as islands of automation or information silos. Also the lack of communication leads to inefficiencies, wherein identical data are stored in multiple locations, or straightforward processes are unable to be automated.

Thus, in a point of view of creating value for the organization it is necessary to understand and apply the concept of Enterprise Application Integration (EAI), which is the key domain of this project.

The main objective of the EAI is the creation of a link between a set of computer enterprise applications, promoting the basic principles in business such as communication, cooperation and coordination. Of course, this process of integration only exists if the various systems have a common business model to ensure a syntactic and semantic unification between the information flow and shared knowledge.

The retail world, where this project will take place, is also an example of the described scenario, taking into account the continuous efforts to improve its Enterprise Resource Planning (ERP) market leader applications, which in this case aim to optimize the retail business processes.

However, given the constant mutations of a market as dynamic as the retail, the proposed EAI project should implement a flexible strategy, offering the ability of diverse systems within the organizations to work together. This can be achieved using a Service-Oriented Architecture

(SOA), a system architecture in which application components are built as interoperable and reusable services.

One of the main advantages of the SOA architectural style is the fact that the developed business services are technology-neutral and given the reusability aspect of these services enterprises can simplify and accelerate new business applications development and deployment.

Therefore, by implementing SOA, a retail organization can align people, processes and data through consolidated applications and shared information services, by orchestrating consistent data sources across previously incompatible systems. With a strategy like this retailers can harvest better sales data, strengthen loss prevention, reduce in-store pricing errors, improve promotion personalization and timing and deliver more in-depth product information directly to customers and sales associates.

With SOA, retailers are squeezing new efficiencies from existing processes and systems, which can be key determiners of profit in today's competitive environment.

1.1.1 Institution of Project: Wipro Retail

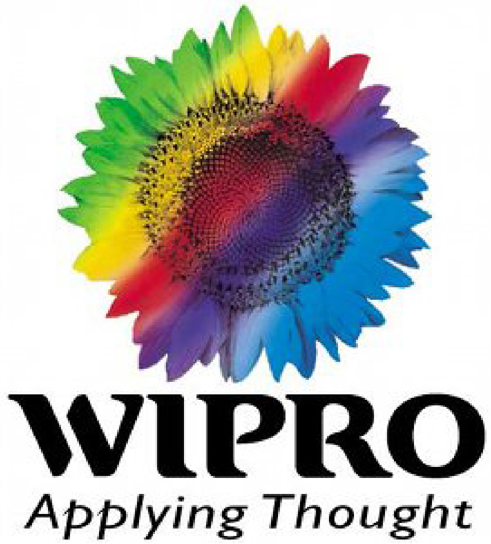


Figure 1.1: Wipro logo

Wipro Retail, a division of Wipro Technologies, based in Bangalore, India, is a global services provider focused on becoming a leading specialist in the implementation of integrated solutions and effective support of retail systems. The company's reputation is based on consistent, high quality service and a track record of delivering excellent results, by implementing well engineered solution delivered to the highest standard, on time and on budget. In fact, Wipro's software development methodologies are the state of the art in the software engineering, being the first company in the world to own a CMMI level 5 certification [WRP08].

Wipro Retail, was formerly Enabler, a company founded in 1997, created through the planned separation of the IS/IT department of Portugal's leading retailer, Modelo Continente, a division of Sonae Group.

Following the creation in Portugal, Wipro Retail has grown its business into a truly international success; having now offices and operations in the major European countries. Furthermore, the expansion continued beyond Europe, by acquiring costumers in North America, Latin America, Middle East, Africa and Asia Pacific.

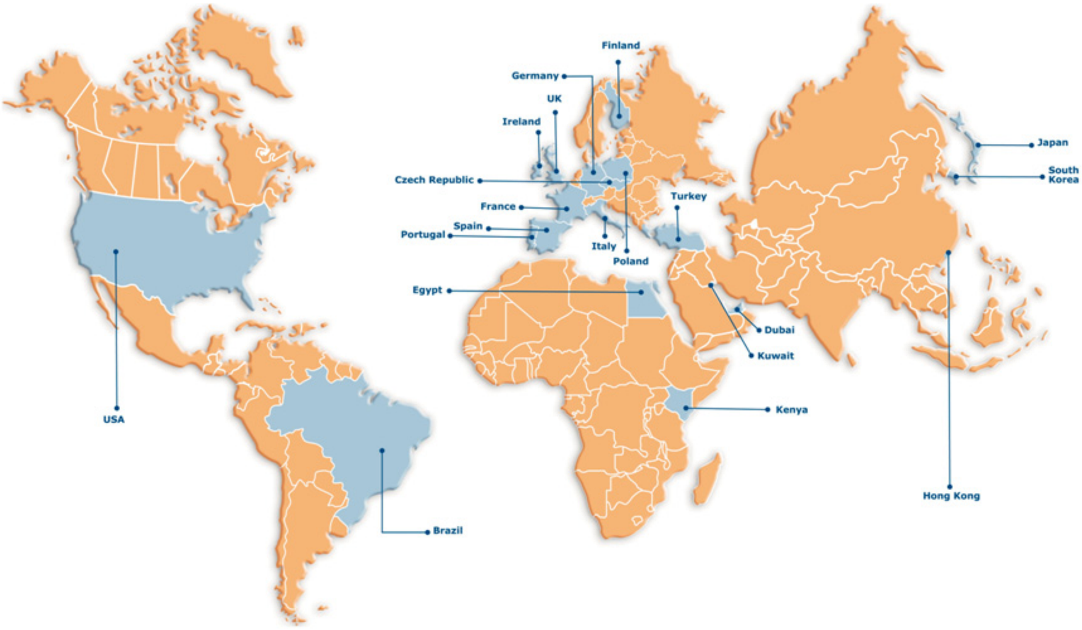


Figure 1.2: Wipro Retail offices and operations

Since the year of foundation, Enabler and presently Wipro Retail, managed significant and sustainable growth, focusing on leading retailers, such as AVA, Modelo Continente, Morrisons, Nisa Today's, Espirit, Despar, Renner, Galeries Lafayette, Sabeco e Fortress.

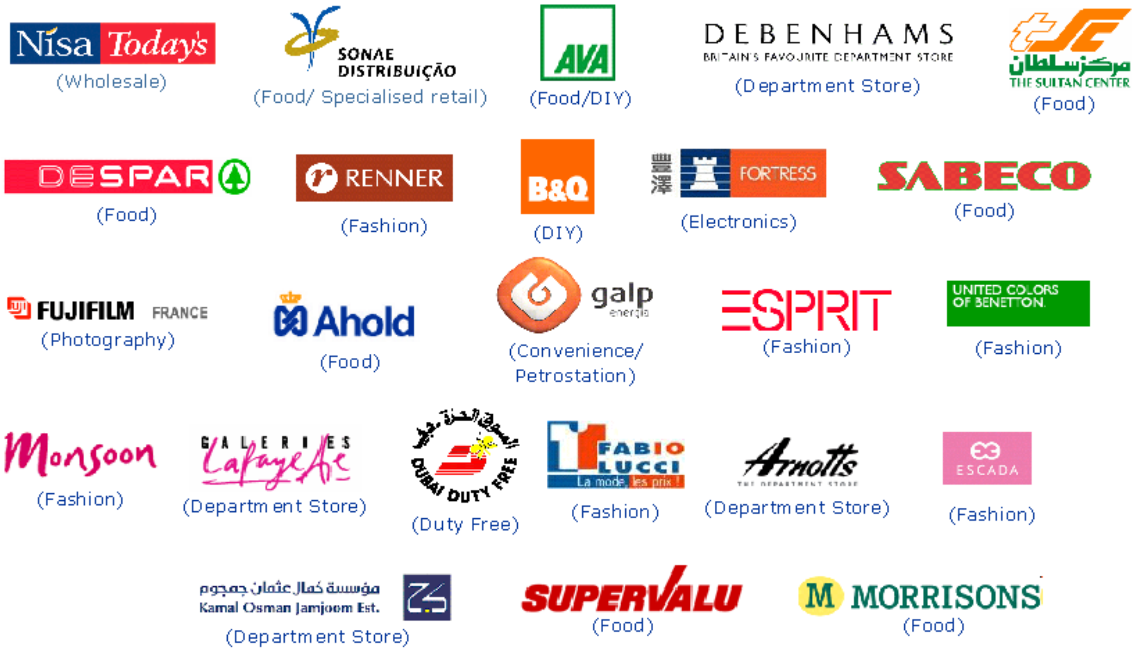


Figure 1.3: Wipro Retail clients

On the delivery perspective, Wipro Retail is organised in competency centres. Each one of these structures will manage, maintain and develop a specific expertise related with the retail business.

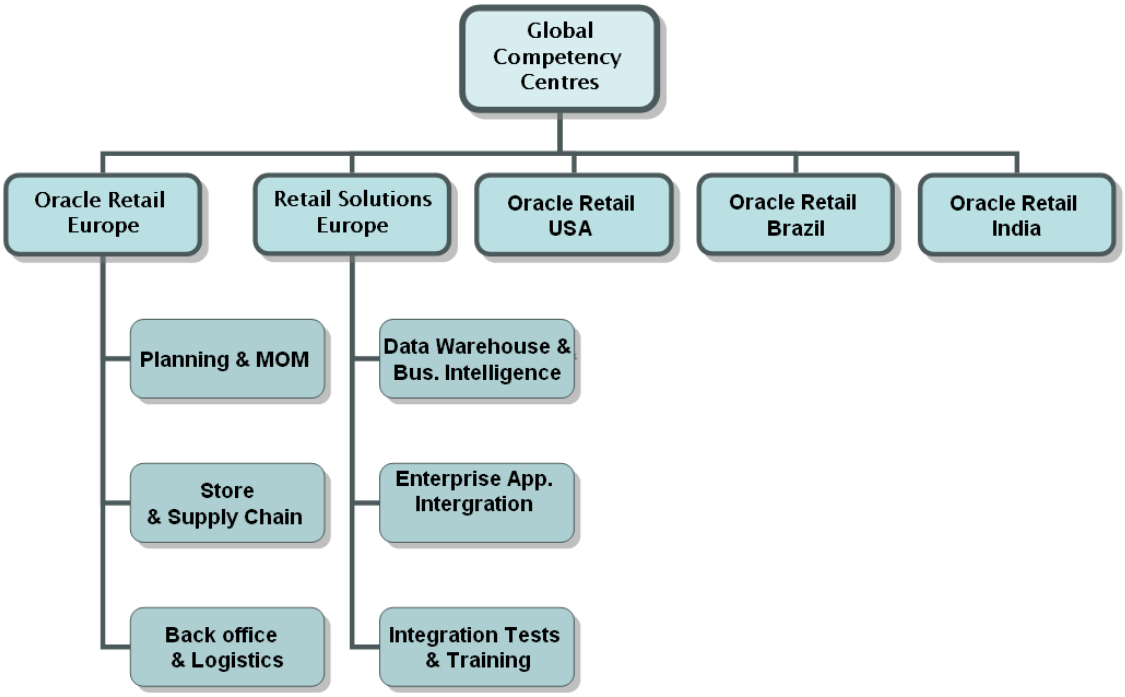


Figure 1.4: Wipro Retail competency centres

The project described in this document represents the implementation of a retail solution in the Enterprise Application Integration (EAI) area of competency.

1.1.2 The Client: DeSpar

DeSpar is a distinguished brand of Italian food-retailing, which belongs to the Spar group, the largest consortium of retailers in the world by store count, with nearly 13,700 stores in 33 countries worldwide [Des09].

Spar was founded in the Netherlands in 1932 by Adriaan van Well, as a voluntary chain of grocers under its original name "DeSpar", the acronym for the Dutch sentence "Door Eendrachtig Samenwerken Profiteren Allen Regelmatig" which translates into "We all benefit from joint cooperation" [SPAR09]. As the organization expanded across Europe, the name was later abbreviated by dropping the "De", except in Italy, which is the only worldwide store that maintains its original designation.

Van Well used the symbol of the fir tree, "Spar" in Dutch, to identify the organization and it became the Spar logo. DeSpar, in order to keep with the international branding, maintains the "Spar" section of the logo highlighted.

Figure 1.5: Spar and DeSpar logos

Since the date of its foundation until now, the Spar group is referred as a wining concept. A testimony to this successful concept are the many awards and accolades all the different Spar countries have won, thus always developing new business, ensuring that Spar is the world's largest retail food store chain.

The link between DeSpar and Wipro Retail (Enabler at the time) dates back to 1999 [PPCS09], when the two companies have signed-off a proposal for the implementation of retail solutions proprietary of Oracle: Oracle Retail Merchandising System (ORMS), Oracle Retail Distribution Management (ORDM) and Oracle Retail Data Warehouse (ORDW). This was the first international project of Enabler. Since the first go live, which was for the ORMS implementation project, on January 2000, Enabler and now Wipro Retail has been supporting DeSpar batch and applications. In the subsequent years, deals for Application Support and Maintenance (ASM) have been signed, with the last contract signed in 2008 for the next 2 years (2009 and 2010), from which this project is included.

1.2 Project

Following a demand for standardization of the systems within the Austria Spar International AG (ASPIAG), an international subsidiary of Spar's Austrian group, the DeSpar stores decided to implement the IMAge (Integrated Management Application for Grocery Enterprises), a solution that provides a browser-based interface which seamlessly combines information and functionality from local and centralized sources [Int08]. This system falls into the concept of an Enterprise Resource Planning (ERP), and can be basically described as a solution that offers features such as inventory control and ordering, along with data analysis capabilities.

Since DeSpar has been using ORMS as its ERP since the year 2000, whose implementation was carried out by Wipro Retail, this situation creates a need for integration between the ORMS and this new ERP, the IMAge.

Given the fact that the IMAge represents a pilot-project, a complete and immediate migration of data should be avoided, thus ensuring no loss of information and guaranteeing that the needed time is given to the users of this new ERP so they can adapt to its new features and especially the business processes that the IMAge contemplates. By doing a phased migration, a more reliable integration between these disparate systems can be achieved, as well as a real-time and fast integration, which are the main goals of the Enterprise Application Integration (EAI).

There are innumerous ways of integration and one of the most recommended and currently used in the client is the Publication/Subscription model, an integration pattern based on an asynchronous messaging paradigm where one or more applications send messages (publishers) into a staging area, frequently denominated as integration BUS, and one or more application receive them (subscribers). The reasons for this being a highly recommended integration solution is based on advantages such as application decoupling, transparency and scalability.

The tools and applications available on the market are also innumerous, being Sun Microsystems one of the main competitors. Sun's main and most recent EAI platform is known as Java Composite Application Platform Suite (JCAPS), currently in its version 6, provides a standards-based, open, extensible platform for developing software infrastructures using a Service-Oriented Architecture (SOA) approach [Sun09b].

So, in general terms, the main objective of this project is the creation of data integration interfaces between the ORMS and the proprietary system of DeSpar, the IMAge, using a Publisher/Subscriber approach with Sun JCAPS 6 as main technology. With this tool it is possible to go much further than a vanilla implementation (a software that is used without any customization applied to it), which is necessary whenever the applications to integrate come from different software companies, as in this case.

A project of this kind is essential to DeSpar, since it inherits all the advantages of a SOA architecture, using EAI strategies, which have associated an extremely flexible data communication model, that in addition to respond to new business processes implemented by the IMAge, also ensures a real-time communication between this ERP and the ORMS.

1.3 Motivation and Objectives

The main motivation for choosing this project is supported by a personal interest by the author on every work involving the most recent distribution and integration technologies. This interest has been developed during the author's academic training, always being an area of preference in terms of future work.

Also, the fact of having already some experience on the EAI area, using a SOA approach, and knowing in advance that this architecture allows for a more quick development and deployment of applications, to meet today's rapidly changing business needs, is also a factor of enthusiasm for participating in such a project. In addition, working with a top level architecture, that goes beyond the traditional object-oriented architectures, therefore giving the opportunity of having disparate systems choreographed into a far more flexible composite application.

Furthermore, the integration of Wipro's EAI team and having the possibility to work for the largest retailer in the world, in order to deliver a vital solution on the point of view of creating value for the company, since the various organizational systems must be integrated to guarantee the flow of business concepts.

Regarding the main goals of the project, at the end of the internship should have been achieved the following objectives:

- Develop team work capabilities
- Acquire and understand the processes associated with the development of solutions for EAI in the context of the retail world
- Develop skills to implement infrastructures based on a SOA approach, using the Publication/Subscription model
- Increase competencies in distributed computing
- Develop knowledge in Messaging Oriented Middleware (MOM)
- Familiarize with JCAPS 6, exploring the following concepts:
    - Business processes, using Business Process Execution Language (BPEL)
    - Collaborations
    - Connectivity maps
    - Deployment profiles

1.4 Structure of Dissertation

Besides the introduction chapter, this document is organized as follows.

In chapter 2 is presented a business overview that characterizes the domain of the project, the retail world.

The chapter 3 details the technologies and development techniques used throughout the internship. It is intended with this section to describe the problematic of the enterprise integration, the main subject of this project. The technologies involved with the developed solution are also discussed in this section.

In chapter 4 a detailed presentation of the problem to solve is made as well as the work developed to implement the integration solutions, according to an EAI approach.

Subsequently, in chapter 5 the testing strategy is explained, detailing the different test approaches used to verify and validate the project developments.

Finally, in chapter 6 the report of the project is concluded, evaluating the given results and pointing out near future work.

2 Business Overview

In this section is presented a short functional overview of the business which characterizes the domain of the project, the retail world. Firstly, it starts by describing the business of retail, allowing the reader to retain some basic notions about this business model. Subsequently, are detailed the types of retailers, the retail segments and their factors of differentiation.

2.1 Retail

Given the scope of this project, it is essential to acquire a notion of the business and all the surrounding retail concepts, since these will always be present during the internship.

The business of retail consists in the distribution and sale of goods for final consumption. This is a particular business, characterized in terms of IT as the handling of large volumes of information, involving thousands of items and transactions.

Today, the retail is a dynamic and complex sector, which includes all the activities of purchase of goods intended for resale or to final consumers.

Markets suffer constant changes and competition is increasingly tight, thus creating a strong competition among retailers. The increasing demand of customers leads to higher levels of quality in services. With the increasing globalization of markets, also in the retail sector there is an increasingly global consolidation and greater number of multinational retailers, therefore preferring more robust and scalable solutions. In addition to being subject to market conditions, retailers also depend on the ability to deal and understand their customers. It is essential to establish relationships of trust with them, and won their loyalty by the quality of services provided. Maximizing the return of investment by retailers, largely depends on operating costs (effort reduction without compromising the quality of services) and the ability to negotiate with the suppliers.

In the retail industry, the main function is the distribution, which includes the processes necessary for transaction of goods. The distribution includes the transportation, receipt, storage, transfers, returns and deliveries. The chain attached to the distribution begins on the suppliers, after orders from retailers and wholesalers. For this purpose, a supplier-retailer interaction is needed, which can be made via EDI (Electronic Data Interchange) or simply by fax. This electronic communication aims to facilitate and accelerate the process of information exchange.

The basic element for the retail business is the article for which each case is a case. Articles may be consumable in short periods of time, require specific conditions of transport and supply and have high turnover rates or simply seasonal, so the management of stocks should be optimized. For this reason, the retailer should at any point in time know the status of items under it, taking note of prices and actual quantities. The access to a prices history is important to boost promotions or mediate future negotiations with the suppliers.

The strategy to be applied by retailers depends largely on market conditions, but also of future consumers. This implies a more meticulous and careful observation of them. To ensure a more personalized service is important to know its recipient. The analysis of records and information contributes to improved service delivery, to greater loyalty, and in return the retailer may benefit from an increase in sales and sales margins, but essentially winning the trust and loyalty of the customer. Following this, it is more profitable to maintain a client than to start the conquest of a substitute.

To sum up, any retailer has the goal to manage in the best way their suppliers, ensuring proper management of stocks, enhance customer relations, identify and understand the modus operandi of the competition.

Thus, arises the need for IS's in retail to support the management of all information relevant to the business. The integrated management support systems allow greater efficiency in the approach to suppliers, increased flexibility and greater capacity for real time decision making.

2.1.1 Types of Retailers

Retailers can be classified into the three following types:

- **Store retailers** – These market their products in a fixed point of sale. Generally use large areas of products exposure, as well as major media advertising (television, magazines and Internet) to attract customers. Typically sell to the public for personal or household use, but may also sell to corporate customers. Such establishments include office supplies stores, consumer electronics and construction materials. Exhibitions of products samples, petrol stations and services, as well as service companies such as post-sale repair and installation, are considered store retailers.
- **Non-store retailers** – Like store retailers, these are organized to serve the public, but use different methods of sale. The establishments of this type reach customers through methods such as paper or electronic catalogues, selling door-to-door, demonstration in domestic houses, sales in mobile street posts, and distribution by vending machines. Amazon.com, Tupperware, Telesales, are examples of retailers of this type.
- **Corporate organizations** – This type of retailer achieve economies of scale, high power of negotiation, brand awareness and efficient sales force through a purchasing centre for the entire group. Note that even though many stores represent independent brands, an increasing number is now part in some way of corporate organization. Through the concentration of purchases of goods, to buy in large quantities at lower prices, it allows to hire experts to deal with promotional pricing, inventory control and sales forecasts. Sonae Distribuição and franchising brands are examples of this type of retailer.

2.1.2 Retail Segments

The retail segments are groups of elements of particular retail business model that drives different processes and different key performance metric benchmarks. There are 50 niche retail segments. However, 8 key segments can be defined, including the majority of the retailers, as described bellow:

- **Specialty (Softlines/Hardlines)** – Focus on selling a certain type of articles, which are supplied to the customers in a special way, using specialized care services, e.g. Footlocker.
- **Department store** – Distinguish from the other stores by the fact that sell a wide range of products, and do not have a predominant line of goods, e.g. El Corte Inglés.

- **Discount** – These are characterized as low-cost shops which usually offer a lower range of products and services, but at very attractive prices, e.g. Lidl.
- **Drug** – Usually these stores combine prescriptions with convenience by offering broad selections of merchandise in a relatively small space, e.g. M24.
- **Consumer electronics** – Large retailers that focus their activity over a range of goods, thus owning a buying power in that range that allows them to charge prices that the big chains can not compete, e.g. Fnac.
- **Grocery** – The major stores that sell from food to house products and are characterized by practising low prices, with low profit margin, but with a high volume of sales, e.g. Carrefour.
- **Warehouse clubs** – Retail stores, usually selling a wide variety of merchandise, in which customers pay annual membership fees in order to shop. They offer low customer service and keep costs down while operating on low margins, e.g. Makro.
- **Furniture/Home** – These retailers appeal to the more affluent 35 to 55 age group, introducing the concept of showroom as the selling floor. Its supply chain key is to cut the time from the customer's purchase to the home delivery date, e.g. IKEA.

2.1.3 Factors of Differentiation

When envisioning a store concept, a retailer makes decisions on the level of service and range of sellable products, pricing policy, geographic coverage, access to the customer, size and location of the store. It is the balancing between each one of these factors that allow the differentiation among the competitors and the progressive capture of a larger fraction of the possible expense of the power of customers, to generate greater volume of business in their stores.

Subsequently are examined the retailers marketing decisions according to the most important dimensions of a differentiation strategy, which are basically of four types: target market, product range, service levels and price and promotions.

2.1.3.1 Target Market

The choice of the target market is probably the most important decision that a retailer has to make. Until the target market is defined and characterized, a retailer can't make consistent decisions in the range of products, decorating the store, advertising, price or service levels.

In order to ensure that they are reaching and satisfying their target customers, retailers must carry out periodic market surveys.

2.1.3.2 Product Range

This factor must meet the expectations of the purchasing target market in terms of diversity of products and brands within each product, facing the challenge to develop a strategy of differentiated products. Typically, the possibilities are:

- Provide exclusive national brands that are not in sale by competitors
- Present well known brands
- Provide innovative and differentiated products
- Present firstly the latest news in the market
- Offer highly customized services

The adopted strategy will achieve higher levels of customer satisfaction, as well as allow for a policy of purchasing and marketing more focused on customer needs.

2.1.3.3 Service Levels

Another decision to be made by retailers concerns the range of services to offer to customers. These can be grouped in three types, which are described bellow:

- **Pre-sale services** – Include the receipt of orders via phone or mail, advertising, displays or demonstrations of products.
- **Post-sale services** – Consist of order and delivery of purchased items, wrapping gifts, devolutions or installation of products.
- **Additional services** – These include information centres, parking, restaurants, repairs, interior decoration, credit facilities or waiting rooms.

2.1.3.4 Prices and Promotions

The price is a determinant factor and should be decided in accordance with the target market, the range of products and services and competitors. All retailers want to get high sales volumes and higher margins. However, most retailers either have large margins and low sales or low margins and high sales volume.

Regarding promotions, retailers use various means to increase sales. From advertising, discount coupons, reward programs or product samples, retailers have to choose the most appropriate means of promotion to support and strengthen its market position.

3 Technical Review

As its own designation indicates, this chapter conducts a review to the main technologies and methodologies used in the project. It is identified, in this section, the main problem addressed by the dissertation, the enterprise integration. The architectures and technologies involved with the developed solution are also detailed in this section.

3.1 Enterprise Application Integration

Enterprise Application Integration (EAI) is defined as the process of linking applications and enterprise data sources so that they can easily share business processes and data. This process must be accomplished without requiring significant changes to these existing applications and the data [UTPG08].

Before EAI, the task of integrating enterprise computer applications and data within a corporate environment has been an expensive and risky proposition, mainly because companies were trying to combine applications that often ran on different hardware platforms and had no protocols for communicating with other software packages outside of their own narrowly defined realm. In a sense, companies had “islands” of business functions and data, and each island existed in its own separate problem domain.

To solve this problem, following an EAI approach, it is necessary to define semantics for application and data integration. That is, EAI defines a standard methodology or approach for applications and data sources to communicate. By supporting this standard, applications can easily communicate with other applications and data sources. The pieces in the integration puzzle – such as an underlying database management system (DBMS) – can change, but because of this common methodology, the replacement piece can be plugged in and the communication can continue uninterrupted.

Thus, the EAI is best suited for heterogeneous environments, which describe a point reached by a certain company due to some reasons, such as acquisitions or mergers with other companies, where they have been compelled to absorb some other company’s systems into their own environment. They may have been trying to increase their capacity – or avoiding replacing existing systems – by patching their own internally developed systems or other purchased systems onto their core systems. Or, they may be supporting large numbers of users on distributed systems with a multitude of platforms.

In short, EAI allows the enterprise applications to work together, thus avoiding that organizations have redundant activities, higher costs, and inefficient response to their customers.

3.1.1 EAI Topologies

The topology of an EAI defines the structure in which each application communicates directly with the other applications. Therefore, when implementing EAI, there are two choices that can be made: point-to-point or bus-centric.

The point-to-point model describes a decentralized structure in which each application communicates directly with the other applications. This type of integration is most appropriate for organizations that need to integrate few applications with a small number of services. Figure 3.1 illustrates the number of connections required for a sample point-to-point environment.

In a point-to-point integration environment, interfaces between the applications are usually written as business needs arise. The problem with this approach is the lack of consistency, as well as the need for a reengineering whenever new business goals are defined that require applications to communicate with one another differently. Every change makes the environment more difficult to understand, until eventually the structure is so complex that it is almost impossible to manage effectively. In fact, such a complex structure doesn't carry benefits to the company due to the high IT costs surrounding the change requests.

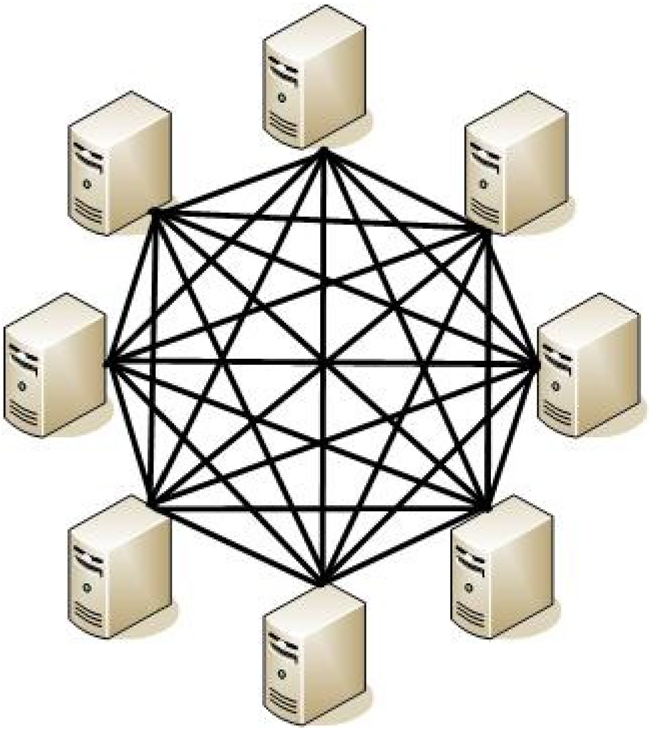


Figure 3.1: Point-to-point topology

The bus-centric model provides a more centralized structure, in which an integration bus is placed between the applications, and each application communicates with the bus rather than communicating directly with other applications. This can be seen in Figure 3.2 which shows a bus-centric topology where applications are connected through an integration bus.

Each application needs only an interface and a connection to the integration bus. To simplify matters, the integration bus can rely heavily on existing standards, which means that either the interfaces that already exist or the methodologies for writing them are well-defined.

The main advantage of a bus-centric approach is scalability. Imagining a typical large-scale organization with thousands of islands of information, involving thousands of applications, it wouldn't be reasonable to create individual interfaces for every point of interaction. Instead, the solution is to create an application integration environment that allows all of the applications to communicate in a logical, predefined way. This bus infrastructure allows to modify or update elements much more easily, and to do so when the business requires, rather than when the preexisting technology dictates. It should also allow the organization to more easily change direction and to use the products and services it has to match evolving requirements.

Because interfaces are within the integration bus (and are usually standards-based), there is no need to rewrite them whenever new applications are introduced. However, the bus can be technically challenging to implement and may be too costly for more simple application integration environments. In addition, some data complexity may have to be sacrificed to ensure that each application conforms to the standards of the integration bus.

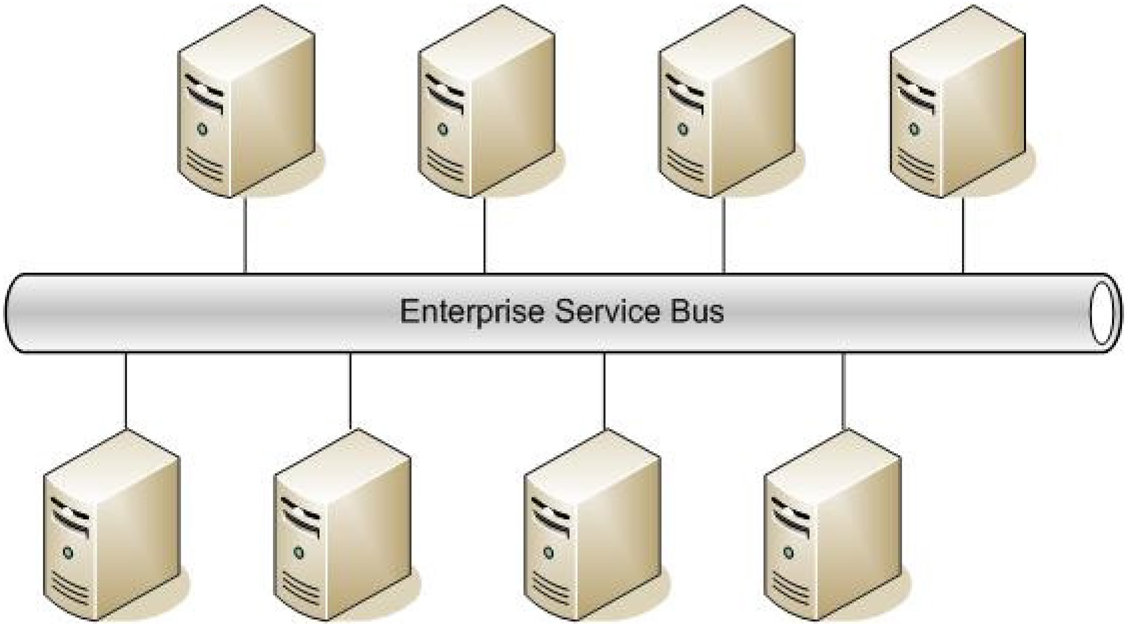


Figure 3.2: Bus-centric topology

3.1.2 Types of EAI

Depending on the processes and data which require integration, EAI can be of the following types: data level, application interface level, method level and user interface level.

3.1.2.1 Data Level

It is the process of moving data between data stores, which can be extracting information from one database, processing that information as required and then updating in another database.

The main advantage of the data level EAI is its relative simplicity. Though it may sound simple, it is actually not very simple as one has to understand the complexities associated with the database as well as how the information flows.

3.1.2.2 Application Interface Level

It refers to the leveraging of interfaces exposed by custom or packaged applications. Using these interfaces, developers are able to bundle many applications together, allowing them to share business logic and information. In order to integrate such systems with other systems in the enterprise, the information must be extracted, place it in a format easily understood by the target application and then transmit the information.

3.1.2.3 Method Level

Consists in the sharing of the business logic that may exist within the enterprise, for example, a method for updating a customer record may be accessed from any number of applications, and applications may access each others methods without having to rewrite each method within the respective application.

Method level EAI allows the enterprise to be integrated through the sharing of common business logic or methods. This can be accomplished by either defining methods that can be shared or by providing the infrastructure for method sharing. Methods may be shared either by hosting them on a central server or by accessing them between applications (e.g. distributed objects).

3.1.2.4 User Interface Level

This is a more primitive but nonetheless necessary approach. Using this scenario, architects and developers are able to bundle applications by using their user interfaces as a common point of integration. For example mainframe applications that do not provide database level access may be accessed through the user interface of the application.

Though the other EAI levels are much more appealing technically, in many cases the user interface is the only available mechanism for accessing data and logic. In spite of the inefficiencies the advantage user interface has over the other is that it does not require much changes to the source or target systems.

3.1.3 EAI Architectures

The following sub-sections give the reader a brief description of the integration architectures used in this project.

3.1.3.1 Service-Oriented Architecture

Service-Oriented Architecture (SOA) is a newer architectural style of distributed computing. It is based on creating loosely-coupled, independent, reusable business software services that are interoperable, technology-neutral, and that can be choreographed or orchestrated into a composite application running on the Internet. It can support a transformed or modernized business process workflow and integrate partners, customers and suppliers. Since business software services are reusable, new applications can be built more quickly [ISM08].

The mentioned business services can be created from existing software assets and packaged applications, which grant to SOA a much greater business, IT flexibility and adaptability, because composite applications can easily be changed and adapted. The integration is completely automated by the architecture. Changing SOA-based applications to reflect future business change is thus dramatically easier and faster. Also, within SOA services are loosely coupled which means that they don’t have to know the technical details about each other to be able to communicate, which is the single most important characteristic of SOA. This guarantees the autonomy of services, allowing for them to be developed, installed, and maintained independent from the other services.

SOA is based on, but extends, web services technology and exploits the broad, maturing set of open web services standards needed for large-scale deployment. These enable SOA to provide seamless interoperation of web services based business software services to run smoothly across different hardware, operating system and middleware platforms, and across different programming models, over the Internet, under open standards.

Thus, the benefits of SOA are directly related to the characteristics of the services that this architecture implements. A service can be defined as a reusable function that can be invoked by another component through a well-defined interface, with the main role of exposing important business services in a flexible, easily composed and highly reusable fashion.

So, the most important characteristics of a service are:

- **Loose coupling** – Services hide implementation details and minimize dependencies with the requesting party.
- **Reusability** – Services must be reusable by definition. Services can be reusable over several lines of business.
- **Composition** – Services can be assembled and coordinated to form new services.
- **Aggregation** – Services can be aggregated to form new functions. Aggregated functions are more specific and therefore less reusable than the services they are composed from.
- **Generality** – Services can be specific to a particular process or channel, or they can be generic in nature. Generic services are more reusable.
- **Granularity** – Granularity is defined by the amount of data a service processes or by the amount of processing in term of functionality the service implements. Granularity is one of the major design issues in the service design process.

Since, with SOA, the application components are built as services that inherit these characteristics, this allows offering a set of IT components flexible in both interoperability and reusability, which can be used to support composite business processes, contributing for saving costs within the company.

3.1.3.2 Message-Oriented Middleware

Message-Oriented Middleware (MOM) is a client/server infrastructure that increases the interoperability, portability and flexibility of an application by allowing the application to be distributed over multiple heterogeneous platforms [Sha07]. It reduces the complexity of developing applications that span multiple operating systems and network protocols by insulating the application developer from the details of the various operating systems and network interfaces, by providing an Application Programming Interface (API) that extends across diverse platforms and networks.

MOM, as shown in Figure 3.3, is software that resides in both portions of client/server architecture and typically supports asynchronous calls between the client and server applications. Message queues provide temporary storage when the destination program is busy or not connected. MOM reduces the involvement of application developers with the complexity of the master-slave nature of the client/server mechanism.

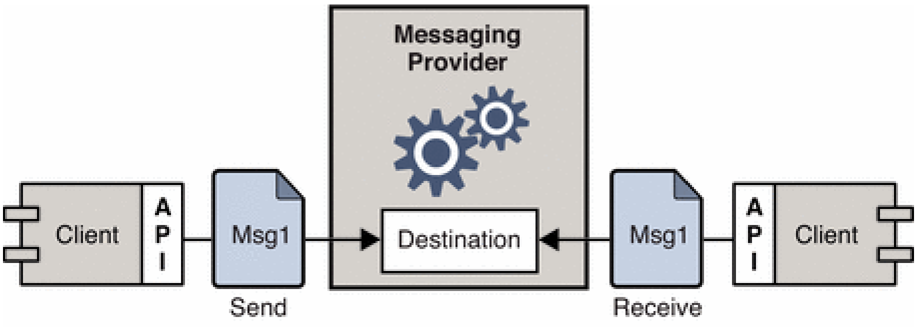


Figure 3.3: Message-Oriented Middleware based system

It also increases the flexibility of the architecture by enabling applications to exchange messages with other programs without having to know what platform or processor the other application resides on within the network.

The mentioned messages can contain formatted data, requests for action or both. Technically, MOM systems provide a message queue between interoperating processes, so if the destination process is busy, the message is held in a temporary storage location until it can be processed.

3.1.3.3 Publish/Subscribe

At a high level, the Publish/Subscribe mechanism helps keep co-operating systems synchronized by one-way propagation of messages because one publisher sends a message to any number of intended subscribers [Nar06].

With Publish/Subscribe messaging, there are message publishers, who produce messages, and message subscribers, who register their interest in particular messages, as illustrated in Figure 3.4. There is also a separate Publish/Subscribe facility that acts as the integration point, where messages are made available by the publishers and delivered to the subscribers.

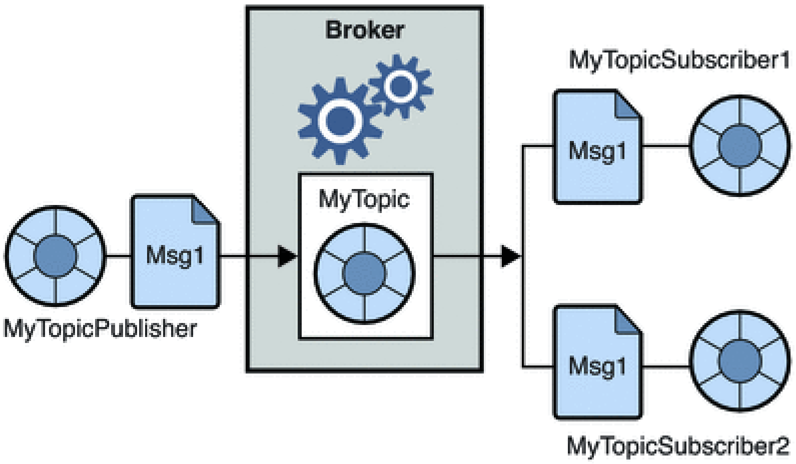


Figure 3.4: Simple Publish/Subscribe messaging

As detailed in Figure 3.4, the Publish/Subscribe pattern works with a publishing application (*MyTopicPublisher*) which publishes a message on a specific topic (*MyTopic*). Multiple subscribing applications (*MyTopicSubscriber1* and *MyTopicSubscriber2*) can subscribe to this topic and receive the messages made available by the publisher. The Publish/Subscribe facility (*Broker*) takes the responsibility of delivering the published messages to the subscribing applications based on the subscribed topic.

Since Publish/Subscribe is typically one part of a larger Message-Oriented Middleware solution, the messaging system provides an API to access the messaging services.

3.1.3.4 Business Process Execution Language (BPEL)

Business Process Execution Language (BPEL) is a XML based language used to define enterprise business processes with in web services. BPEL extends the web services interaction model and enables it to support business transactions. The Processes implemented in BPEL can orchestrate interactions between web services using XML documents in a standardized manner. It has built-in support for asynchronous interactions, flow control and compensating business transactions [Aru08].

In Figure 3.5 is represented a sample business process implemented using BPEL. In this case, the exemplified process illustrates an application publisher and thereby describes tasks associated with the publication process.

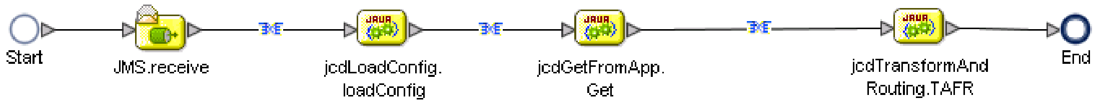


Figure 3.5: BPEL process example in Java CAPS

BPEL supports two different types of business processes:

- **Executable process** – It models the actual behaviour of a participant in a business interaction. They follow the orchestration paradigm and can be executed by an orchestration engine.
- **Abstract process** – This is also a reasonable approach, and it represents partially specified processes that are not intended to be executed. It uses process descriptions that specify the mutually visible message exchange behaviour of each of the parties involved in the protocol without revealing their internal behaviour.

In Java CAPS, the development of business processes can be made in two ways – working directly with the XML source, or using a graphical view, similar to what is shown in Figure 3.5. This BPEL design is based on the Business Process Modeling Notation (BPMN), a standards-based graphical representation for specifying business processes. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation [BPMN09]. The BPMN is maintained by the Object Management Group (OMG), famous by their most-used specification, the Unified Modeling Language (UML).

Thus, the BPEL is emerging as the clear standard for composing multiple synchronous and asynchronous services into collaborative and transactional process flows, facilitating the development of SOA based applications. With BPEL, composite applications can become more agile and adapt to changing business conditions quickly, therefore reducing implementation costs.

3.1.3.5 Database Sharing

This methodology represents a tightly-coupled integration approach where multiple applications share the same database schema, located in a single physical database.

Whilst having to share data between multiple applications each with its own database might be difficult, it is no easier to have multiple applications share one database, because of the difficulty in developing a database schema that satisfies requirements of multiple disparate applications.

There are several approaches to achieve this kind of integration but the most commonly used consists in the development of database packages, with their own procedures, functions and triggers, to allow for CRUD (Create, Read, Update and Delete) operations to be executed directly to the database without requiring external components.

3.2 Oracle Retail

Oracle is the world’s largest business software company, with more than 320.000 customers, representing a variety of sizes and industries in more than 145 countries around the globe [Ora09a].

Although being best-known due to its Database Management Systems (DBMS), the corporation also builds tools for middle-tier software, Enterprise Resource Planning (ERP), Customer Relationship Management (CRM) and Supply Chain Management (SCM).

Following the success of the company and its continuous ambition for expansion across all areas of business, Oracle triggered a market strategy of smaller companies purchase, namely in the area of retail. Amongst these acquisitions, the highlight was the acquisition of Retek, back in April 2005, an important decision in the corporation's history since it strengthened Oracle's position in the retail applications market globally. Retek was the owner of the Retek Merchandising System (RMS), renamed, after the acquisition, to Oracle Retail Merchandising System (ORMS).

Over the years, Oracle increased in size and gained the trust of many customers, now offering an integrated set of modules that include an understanding of the market, with a set of comprehensive tools which support the entire structure of retailing. This set of tools was given the name Oracle Retail.

The modular structure of the Oracle Retail is represented in Figure 3.6. The modules that a given client implements vary greatly depending on their needs and owned legacy systems. From the Oracle Retail integrating modules, only the ORMS will be detailed later in this report, given the fact that was the only module addressed on the developed work.

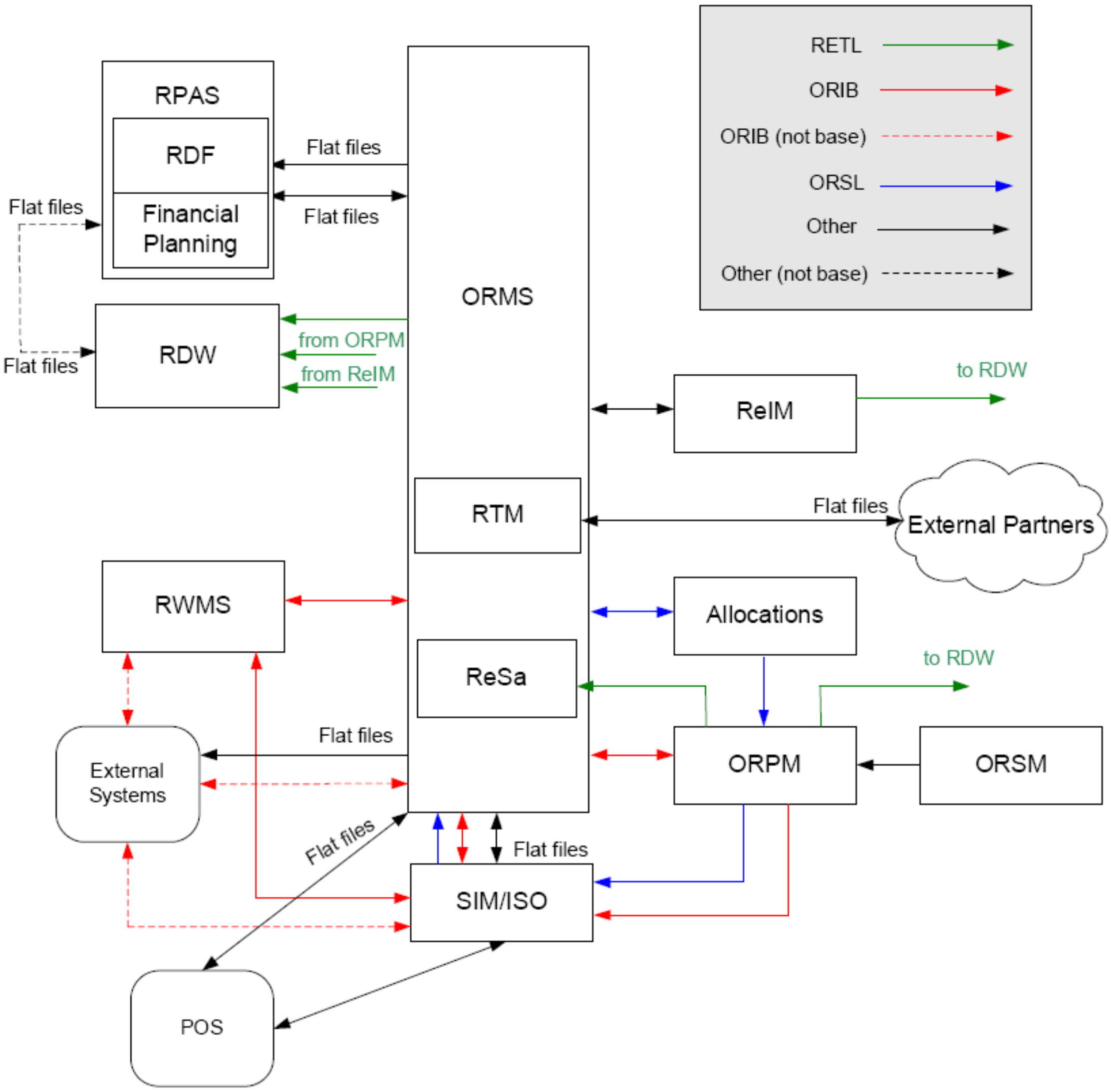


Figure 3.6: Oracle Retail modules

The represented modules are organized into different areas of retail, according to an Oracle Retail solution strategy, with the aim of providing a better profitability of resources and a good supply chain planning. These areas of retail are demonstrated in Figure 3.7 and described afterwards.

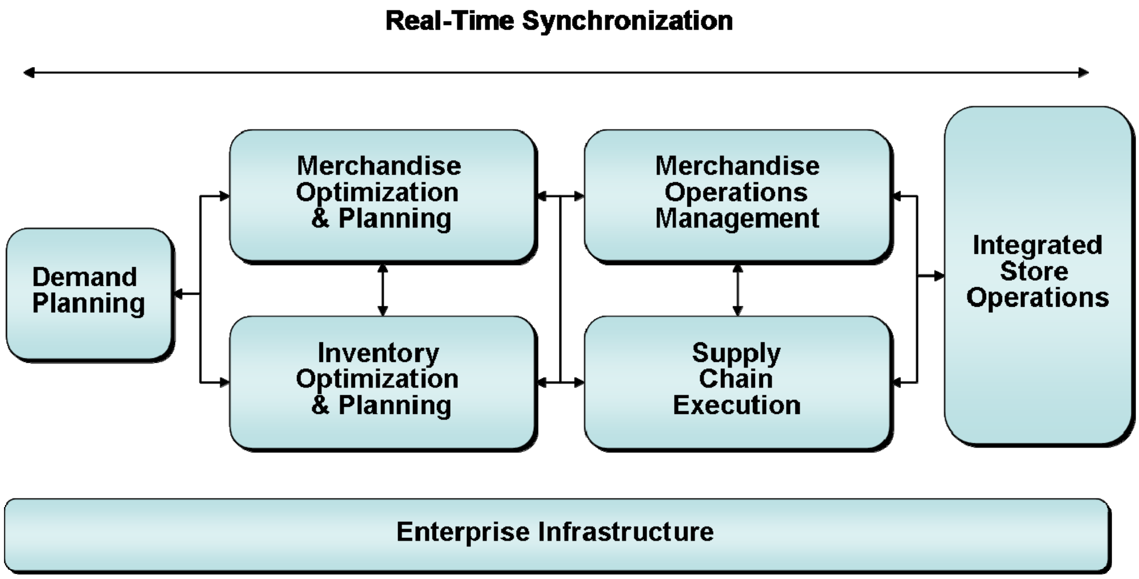

Real-Time Synchronization
Demand Planning
Merchandise Optimization & Planning
Inventory Optimization & Planning
Merchandise Operations Management
Supply Chain Execution
Integrated Store Operations
Enterprise Infrastructure

Figure 3.7: Oracle Retail solution areas

The Oracle Retail business areas are now described:

- **Demand Planning** – This is a powerful and flexible Online Analytical Processing (OLAP) tool, a category of business intelligence tool that supports the forecasting of demand for sales of finished goods. Oracle Demand Planning is part of the Oracle Advanced Planning Solution, and seamlessly integrates with the other Oracle Retail solutions.
- **Merchandise Optimization & Planning** – This set of solutions allow to create intelligent estimates of future business opportunities, providing a common demand planning engine throughout the execution system of the company. The demand planning solutions include a statistical forecast of demand and a forecast of promotions. The products that allow these features are: Oracle Retail Demand Forecasting (ORDF) and Oracle Retail Predictive Application Server (ORPAS).
- **Merchandise Operations Management** – These represent a fully integrated and large expansion set of solutions which can be implemented in an independent manner. Allows the coordination of business operations and maintain a single data source and consistent throughout all the systems and channels. The following products are included in this area: Oracle Retail Merchandising System (ORMS), Oracle Retail Sales Audit (OReSA), Oracle Retail Trade Management (ORTM), Oracle Retail Invoice Matching (OReIM) and Oracle Retail Price Management (ORPM).
- **Inventory Optimization & Planning** – It considers the demand, supply, constraints, and variability in the extended supply chain to optimize the company’s strategic inventory investment decisions. It allows for the organization to provide higher service levels to their customers at a lower total cost.
- **Supply Chain Execution** – The applications in this area are responsible for planning and optimizing the level of replenishment, but also a collaborative inventory management. These solutions allow to make a precise match between supply and demand, which is especially appealing for large product lines, multiple stores and warehouses, complex networks of connection with vendors and environments with large volumes of sales. The system presented as part of these solutions is the Oracle Retail Warehouse Management System (ORWMS).
- **Integrated Store Operations** – These allow for the conduct of the business through multiple channels, thus providing a consistent experience of buying at all channels – stores, websites, catalogues or call centre. The main business activities included in this area involve the management of Point-of-Sale (POS), store efforts management,

customer orders and store inventories. The product that fits in this area is the Oracle Retail Store Inventory Management (ORSIM).

The most important area of the Oracle Retail solutions is definitely the *Merchandise Operations Management*, by the fact it includes the ORMS, the main module of the suite, since it stores and manages all the reference information of the retailer.

3.2.1 Oracle Retail Merchandising System (ORMS)

Oracle Retail Merchandising System (ORMS) is the system that stores and controls virtually all data in the retail enterprise and ensures data integrity across all integrated systems [Goe08b].

ORMS provides easy access to the crucial information of daily merchandising activities and ability to focus on key decisions that help achieve sales and profit targets, by streamlining business practices and unifying business systems across retail channels to better serve customers. Since ORMS has been developed as a web-based, scalable product, it fully supports the large volumes found in retail, leaving time for retailers to concentrate on the bottom line.

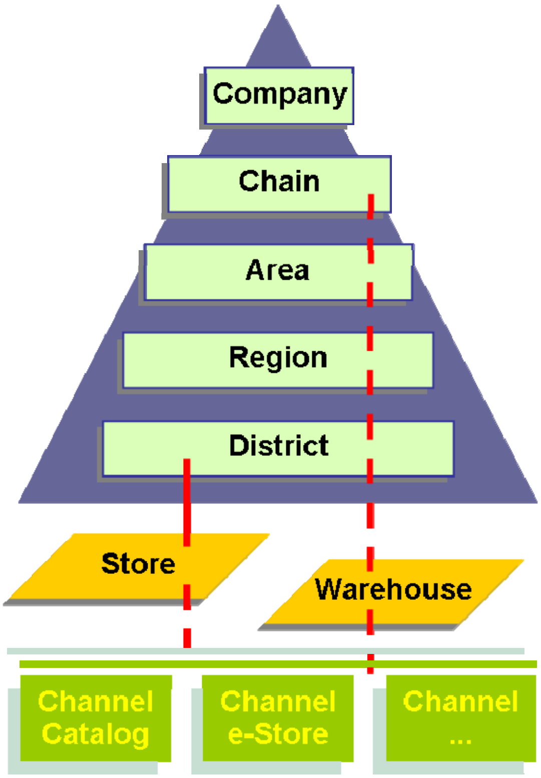
The ORMS key functions are detailed in the following sub-sections.

3.2.1.1 Foundation Data

Foundation data is the data that defines core business concepts, and is shared among all Oracle Retail solutions, such as: products, stores and warehouses, organizational and merchandising hierarchy, suppliers and negotiations.

This data is divided by two structures: organizational hierarchy and merchandise hierarchy.

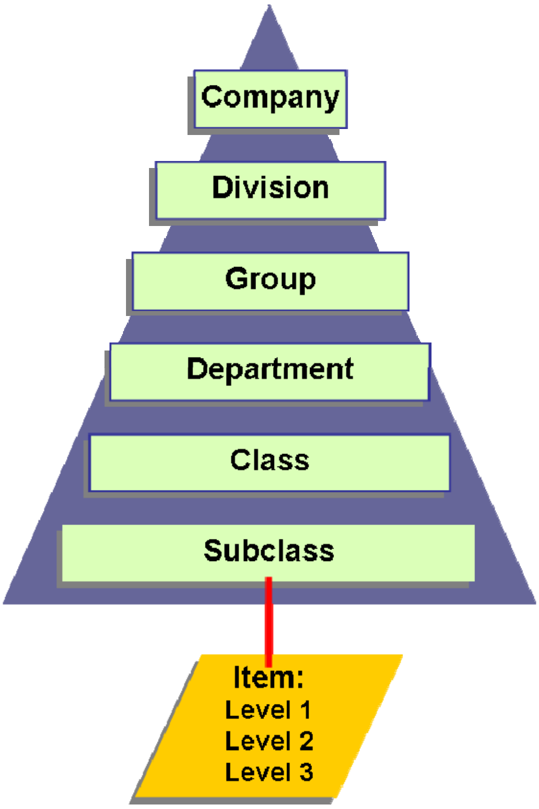
The organizational hierarchy, represented in Figure 3.8, creates the system relationships that are necessary to support the operational structure of a company.



Company – The highest organizational unit defined in ORMS. Only one company can be defined.
Chain – It is used to group various store formats, concepts, and geographical locations within the organization.
Area – Defines a geographical group within the organization. An area can belong to only one chain.
Region – It is also used to group geographical locations. A region can belong to only one area.
District – Represents a geographical location belonging to a given region. A district can belong to only one region.
Store – level where transactions occur, such as sales.
Warehouse – Represents a warehouse. It is not bound to any level of the hierarchy.

Figure 3.8: ORMS organizational hierarchy

The merchandise hierarchy, detailed in Figure 3.9, defines relationships within the system for product management, financial support and productivity report.



Company – The highest merchandise unit, shared from the organizational hierarchy. Only one company can be defined.

Division – Represents the type of goods.

Group – A lower-level type grouping of goods. Belongs only to a division.

Department – It is the level of management of goods, which make reports, budget and inventory and profitability.

Class – Product groups within a department.

Subclass – Grouping the classes into sub-categories.

Item – Level where the articles are kept.

Figure 3.9: ORMS merchandise hierarchy

3.2.1.2 Item Maintenance

The item maintenance process, along with the foundation data, makes up the core of ORMS. Items must be created in the system before the retailer can order, receive or sell any goods.

The items can be of different types:

- **Regular items** – Created by differentiators from common information. The differentiators are related to characteristics of the articles, such as colour, taste or size;
- **Packs** – A group of items that can be sold and/or ordered as one item. There are two types of packs: simple packs (contains multiples of one component item) and complex packs (contains multiple component items).
- **Deposit items** – These items have a portion which is returnable by the customer after it has been sold to him, e.g. Beer bottles.
- **Consignment items** – Represent marketing arrangements where physical control of merchandise is transferred from the consignor (the supplier) to the consignee (the retailer). The owner of the goods remains with the consignor until the goods are sold, e.g. Newspapers.
- **Concession items** – These items are similar to the consignment items in that the retailer does not own the inventory being sold. Concession items differ from consignment in that the ownership is not transferred when the items are sold. A retailer rents floor space to a supplier on which the supplier sells their goods. Concession sales are recorded and the retailer then bills the supplier using their chosen method, e.g. Mobile phones.
- **Transformable items** – Refer to items that can be ordered as one item and then broken into smaller items, e.g. Pork and lamb.

3.2.1.3 Cost Maintenance

The initial cost of an item is established at item set-up at the item/supplier/original country/location level. After an item is approved, any cost changes need to be handled through the cost change form.

Cost changes can only be effected at the transaction level of an item, at three levels:

- Supplier
- Item (and item list)
- Location

These cost changes can be entered into the system via EDI or manually. When approved, the system updates the item supplier cost record for a selected effective date, and can recalculate any outstanding purchase order amounts not yet received.

Cost Changes are applied during the nightly batch on the night before the price change effective date.

3.2.1.4 Deals Management

A deal is an agreement between two parts, fixing benefits and obligations for each other, during a determined period. In the retail business, deals are made (mainly) with suppliers. Most times, it represents a decrease in item cost (a discount) given by the supplier to the retail company. In exchange, the supplier increases his sales and sustains a longer relation with the retailer.

The ORMS supports four different types of deals:

- **Fixed deals** – Used to receive payments from suppliers in return for mentioning their products in promotions or by displaying the product in prime shelf space.
- **Purchase order specific deals** – A deal that is applied to items on a specific purchase order, which supersedes any other deals for the items on that purchase order. The deal is closed automatically when the items on the purchase order are received.
- **Off-invoice deals** – These are used when a cost price reduction is received. The supplier includes the discount in the invoices sent to the retailer. The discount is applied over the purchase order cost price.
- **Bill back deals** – Allow the organization to receive money back after a specific event. Bill backs are calculated based on individual purchase orders or receipts.

3.2.1.5 Purchase Orders

Purchase orders are responsible for maintaining store stock level with merchandise that matches consumer demand. In some cases, the purchase orders can be created by the stores. These orders are sent to the central system for integration and validation, or may be sent directly to the suppliers.

Associated with the management of purchase orders is the process of receiving goods. The receipts of goods in the warehouse or in store are validated on the related purchase orders. Whenever exist discrepancies between the purchase order and the actually received, the system allows the retailer to send the supplier a credit request.

3.2.1.6 Backhaul Allowances

The concept of backhaul is when the retailer collects the goods from the supplier using their own transportation.

Backhaul allowances are used to offset the cost incurred by the retailer in transporting the merchandise from the supplier. On receiving the goods from the supplier, allowances are stored and calculated, and this will affect the stock value.

Backhaul allowances can be applied to the purchase orders in two ways:

- Manually inserted in the purchase order (flat fee)
- Inherited from predefined backhaul allowance (calculated)

3.2.1.7 Inventory Management

The inventory management is relative to the treatment and handling of goods within the retail group.

Transfers are used to move stock within the company from one location to another directly, or if necessary, using an intermediate finisher that will send goods to another location (both physical and virtual locations).

The ORMS transfers can have the following types:

- Returns from stores
- Stock movements between warehouses
- Stock movements between stores
- Stock movements between warehouses and stores due to replenishment or manual transfers

3.2.1.8 Replenishment

Replenishment is an ORMS system responsible for constantly monitoring inventory conditions and based on these conditions create the purchase orders or transfers to fulfil consumer demand.

The process of replenishment is based on indicators located in:

- System options (e.g. purge days, order days)
- Supplier (scaling, defaults for all items/departments)
- Supplier inventory management (review cycle, sups minimums)
- Item/location (replenishment methods, supplier, activate dates)
- Item/supplier/origin country (lead time)

3.2.1.9 Wholesale and Franchising

The wholesale and franchise functionality in ORMS is based on a pull supply chain model, designed to cope with businesses that require working with non-company stores.

Figure 3.10 illustrates the wholesale and franchise business process workflow.

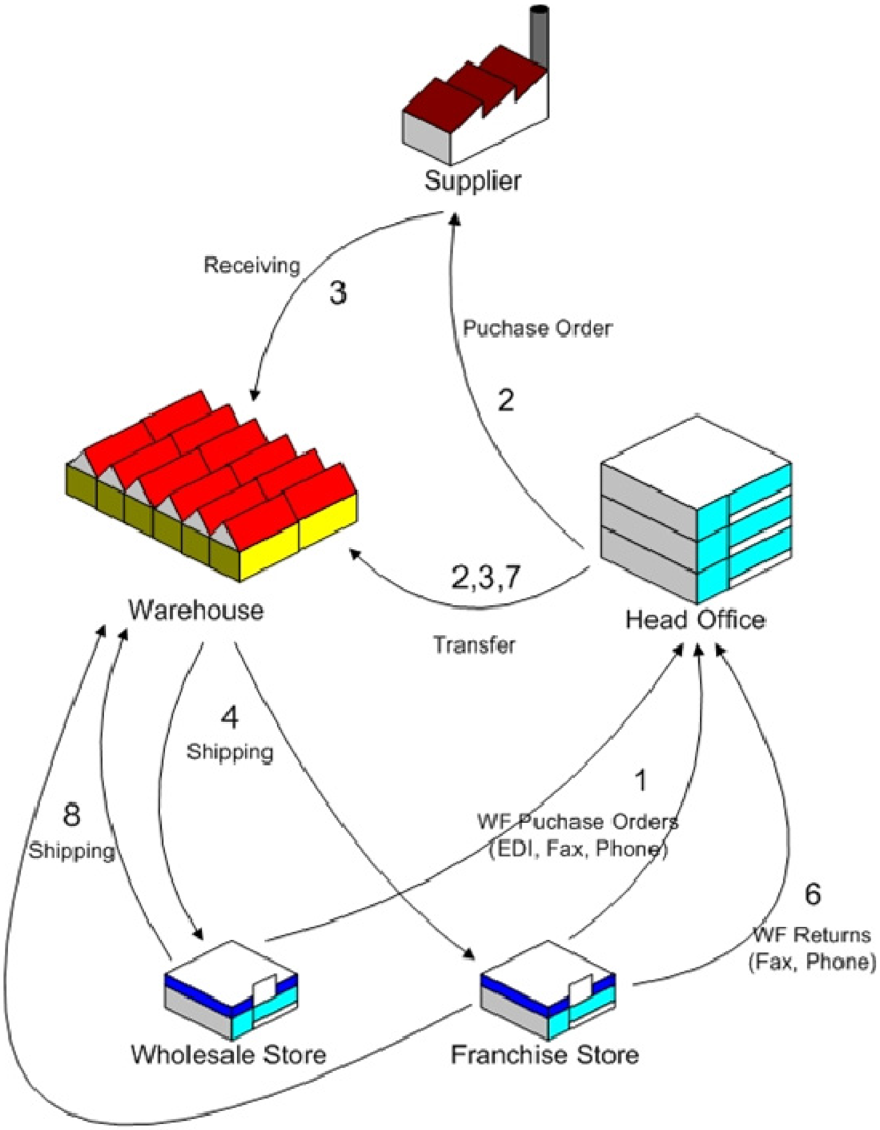


Figure 3.10: Wholesale and franchise business process workflow

The wholesale and franchise stores will send purchase orders via EDI, phone or fax to the head office. Sourcing will always be done through the warehouse (warehouse stocked replenishment) and replenishment will be affected by wholesale and franchise purchase orders, which must be manually approved.

3.2.1.10 Stock Ledger

The stock ledger monitors the actual financial performance of the retailer by incorporating all financial transactions relating to merchandising, including sales, purchase orders, transfers and markdowns. Stock ledger values are summarized by day, week and month, offering a variety of reporting options at the product, location, and time level.

Data is rolled up at the group, division, and company level for profit and loss, balance sheet and retail inventory gross margin percentage reports.

3.3 Java CAPS

On August 25, 2005, Sun Microsystems, Inc. acquired SeeBeyond [Sun08], a software company specialized in systems integration and development of business integration software, particularly famous for its EAI platform known as SeeBeyond ICAN (Integrated Composite Application Network). After the acquisition, SeeBeyond's flagship product ICAN was renamed to Sun Java Composite Application Platform Suite (Java CAPS), or simply JCAPS.

JCAPS provides a standards-based, open, extensible platform for developing software infrastructures using a Service-Oriented Architecture (SOA) approach. This unified and comprehensive suite can help companies create composite applications from existing investments as well as deliver new business services in a flexible SOA environment with no vendor lock-in [Sun09b].

JCAPS, currently in its version 6, provides the needed tools for designing, deploying, and managing platform-independent vendor-neutral composite applications. This latest version retains most of the feature set of previous JCAPS releases, originating in the ICAN suite from SeeBeyond, and provides further flexibility through standardization. Among these tools are included integration, infrastructure, and security products as well as powerful developer tools.

The modular architecture of the JCAPS framework is represented in Figure 3.11.

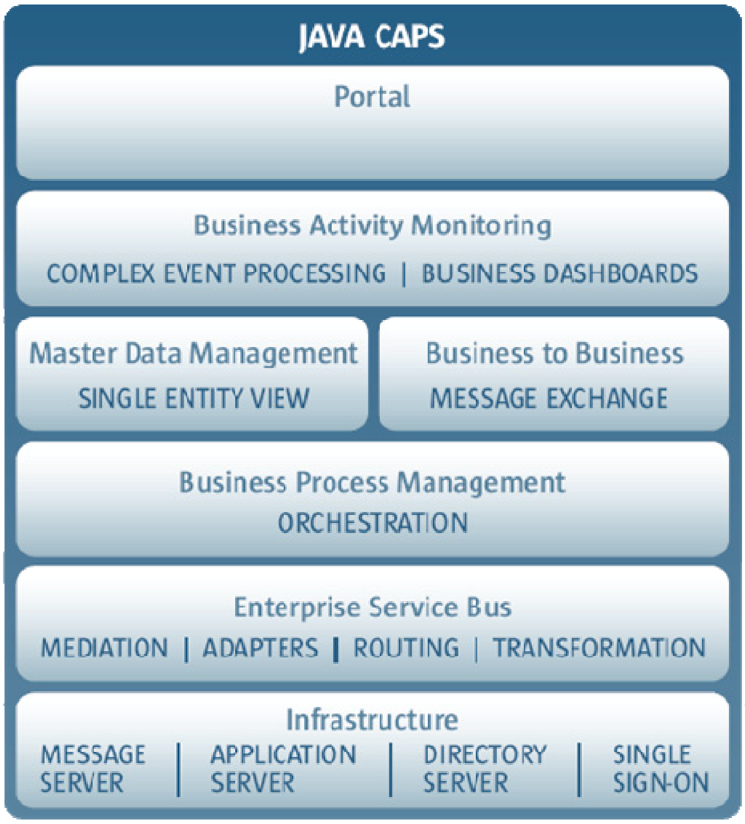


Figure 3.11: Sun Java CAPS architecture

As represented above, the JCAPS framework has a layered architecture, formed with the layers described bellow:

- **Portal** – This layer provides a user portal for collaboration and the ability to personalize content for users based on a user's identity-provisioned account.
- **Business Activity Monitoring** – Enables businesses supervision to determine trends and proactively address critical business issues before problems escalate out of control. Provides an event-driven architecture that supplies real-time trend detection based on a

combination of business event filtering, aggregation, correlation, and analysis across multiple applications and information resources.

- **Business Process Managemen**t – Provides the ability to model, test, implement, monitor, manage, and optimize business processes that orchestrate the flow of activities across any number of web services, systems, people, and partners. Sun's Business Process Manager delivers an open, graphical modelling environment for the industry-standard Business Process Execution Language (BPEL).
- **Enterprise Service Bus** – A Java technology-compliant, web services-based, pluggable integration platform and the foundation of JCAPS. It allows for loosely coupled components to communicate with each other through standards-based messaging, providing a core integration, including comprehensive application connectivity, guaranteed messaging, and robust transformation capabilities, and a unified environment for integration development, deployment, monitoring, and management.
- **Infrastructure** – Provides a strong foundation of application server and identity support for the integration products, as well as a portal interface to support user collaboration with composite applications.

Included with Java CAPS are NetBeans tools, an IDE that provides the single interface for building, testing, and deploying reusable, secure Web services, composite applications, and business processes for Java CAPS. Using NetBeans, the main components to develop for an integration solution are as it follows:

- **Business Processes** – models actual behaviour of a participant in a business interaction, by specifying execution of activities, its inputs/outputs and possible exceptions in the message flow. Business processes in Java CAPS are meant to be made following the Business Process Execution Language (BPEL).
- **Collaborations** – a component that uses Java code to connect to data sources; parse, examine and manipulate incoming messages to form outbound messages; publish outbound messages to data destinations and execute other business logic.
- **Connectivity Maps** – collects logical components, connect them to form message routes, name external systems connectors and, in the case of JMS destinations, specify the types of JMS destination that will be used.
- **Deployment Profiles** – given a collection of external resources, specified by a determined environment, the deployment profile is the mechanism by which logical components of the solution are mapped to physical resources.

In Figure 3.12 is represented a sample implementation of a Java CAPS project for the VAT (Value Added Tax) business entity, illustrating a connectivity map that contains business processes and collaborations. The deployment profile created for this application is demonstrated in Figure 3.13.

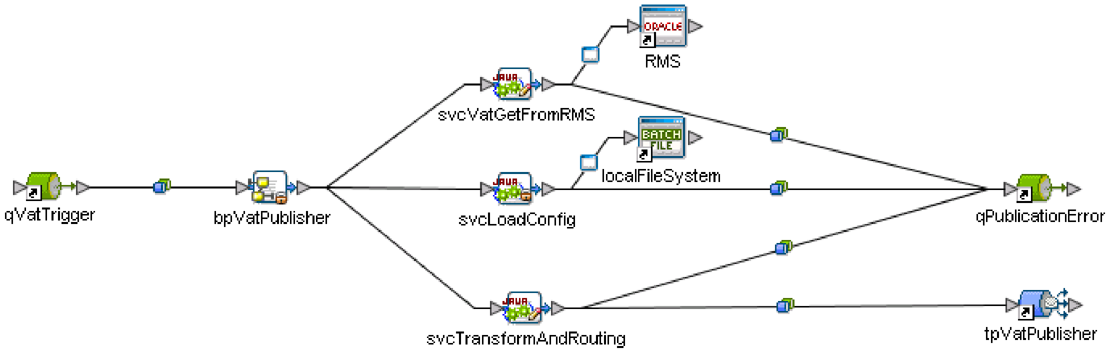


Figure 3.12: Java CAPS connectivity map example

The publication integration flow represented in the connectivity map above is interpreted from left to right. Thus, a JMS queue, *qVatTrigger*, starts the whole process, once a triggering message is received. That message will then be processed by the implemented BPEL, *bpVatPublisher*, which is responsible for executing the following three operations:

- **VatGetFromRMS** – will execute a collaboration responsible for retrieving the VAT family messages from the ORMS application.
- **LoadConfig** – this component will load the necessary configuration parameters available from the properties files of the given Java CAPS project.
- **TransformAndRouting** – applies the necessary transformation and routing operations to the received VAT message. This mechanism never changes the message contents.

After the successful execution of the mentioned BPEL operations, the publication flow is concluded by sending the message to a JMS topic, *tpVatPublisher*. At this point, the VAT data is ready for subscription by any kind of application, as long as it is supported by the Java CAPS adapters. If an error occurs during the publication process, the message is sent to a JMS error queue, *qPublicationError*, being a common practice to have a project subscribing that same queue and sending an error message by email, for instance.

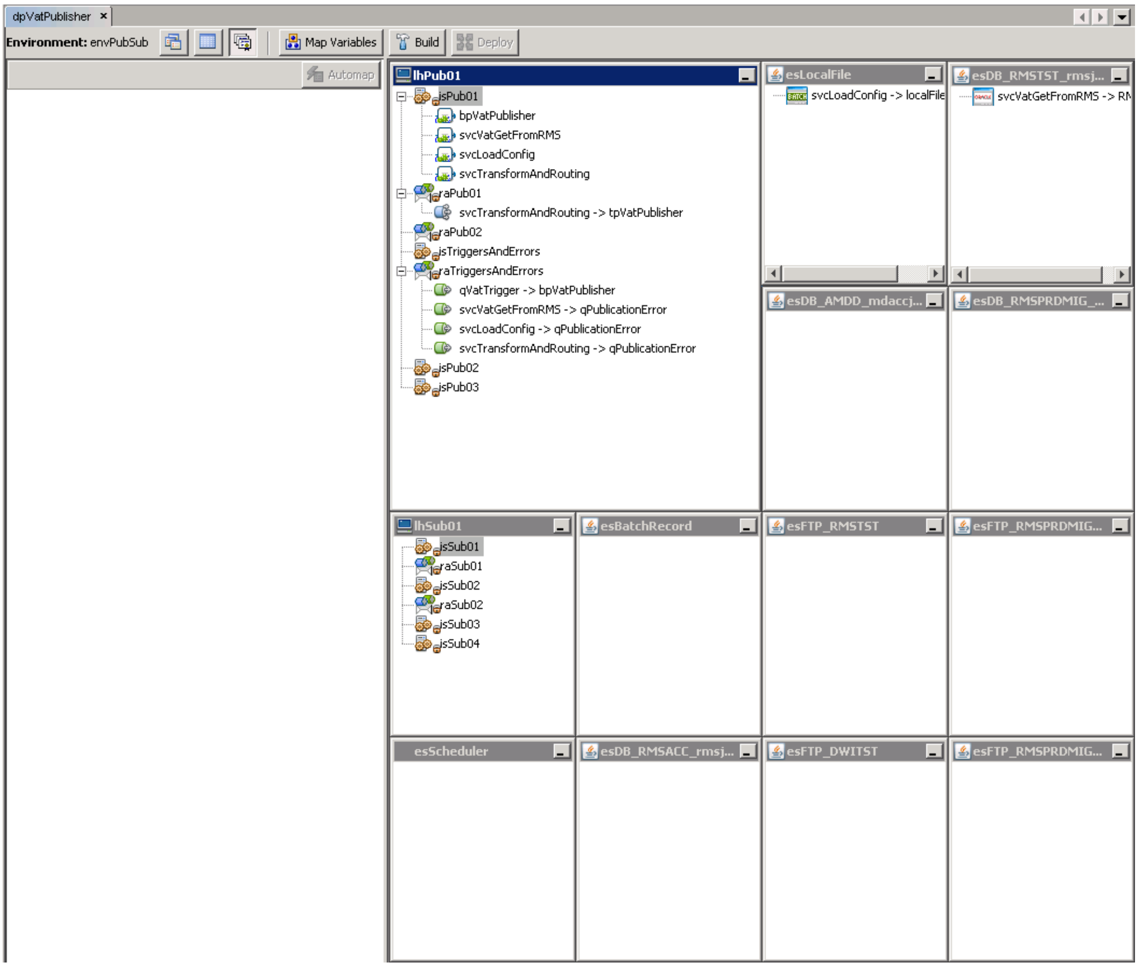


Figure 3.13: Java CAPS deployment profile example

As already described, the deployment profile represented above maps the logical components of the solution into the physical resources. Since this is a publication project example, it is used a Publisher/Subscriber environment and its publication logicalhost, represented in Figure 3.13 as *lhPub01*. After deploying the project, this logicalhost will be in charge of hosting the services and components included in the project's connectivity map.

The interpretation made for this VAT example can be roughly followed for other Publication/Subscription projects implemented using Java CAPS for the NetBeans IDE.

3.4 DeSpar Pub/Sub Framework

In the particular case of this project's client, DeSpar, the implementation of an EAI solution, using JCAPS, will be held with the help of DeSpar Pub/Sub framework. This consists of a bus-centric framework developed by Wipro Retail, using the Java language. The framework development and eventual changes are in accordance with DeSpar needs for integration, adapted to the systems used by the client, the ORMS and the IMAge.

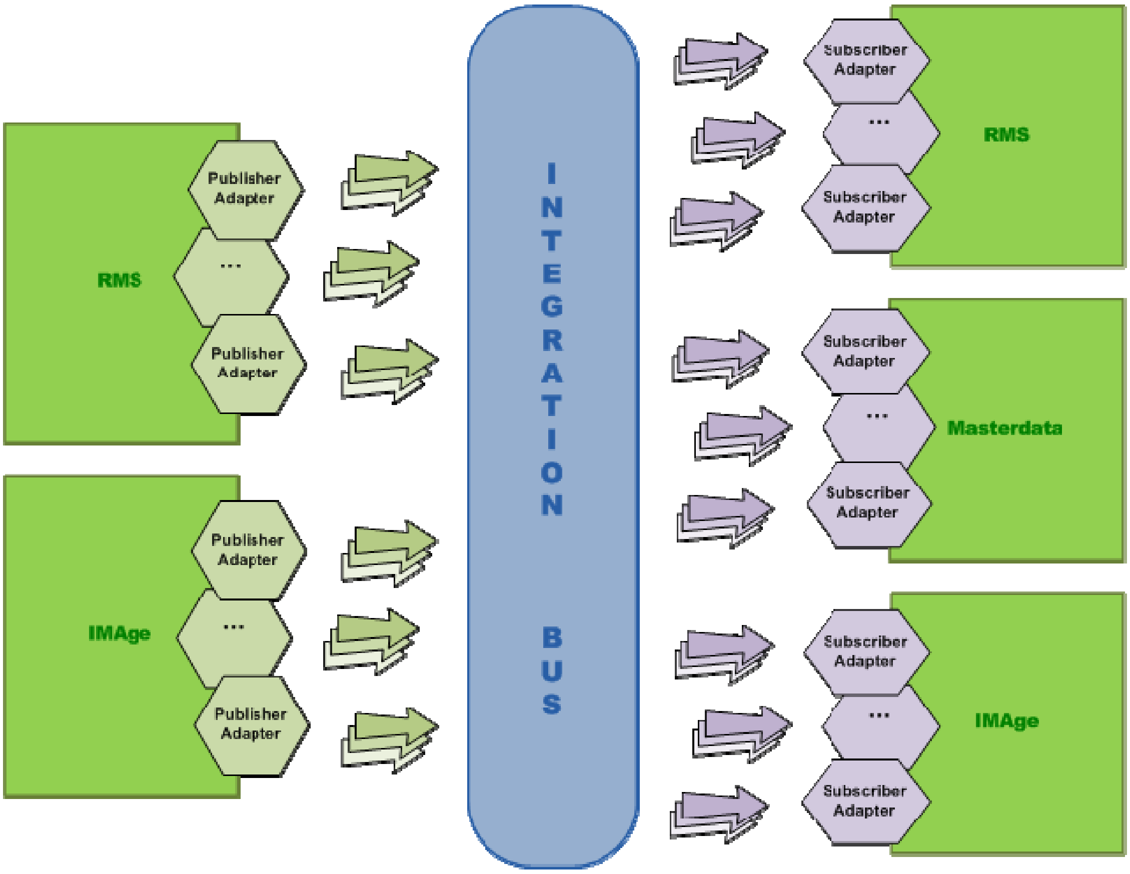


Figure 3.14: DeSpar Pub/Sub framework architecture

In Figure 3.14 are represented the Publisher/Subscriber adapters implemented by the framework and that have been used in this project.

3.5 Conclusions

Since the main objective of this project is not to make an intensive study of the adopted technologies, in order to extrapolate its viability, primarily because it is a project directed to a client, DeSpar, which has been a Wipro Retail customer since 1999, and since the inclusion of

SeeBeyond ICAN and more recently Sun Java CAPS into the DeSpar projects dates back to the year 2007, these are more than proven technologies and in fact demanded by the client.

Despite the factor of the technological restrictions imposed by the client, and since the goal of the EAI is to achieve a reliable, real time and fast integration between disparate systems, then a SOA approach, using the Publication/Subscription model, seems to perfectly fit this scenario.

Characteristics such as loosely-coupled, reusability, modularity and interoperability, which define the ground rules for development, maintenance and usage of SOA, are reflected into business benefits, since by allowing faster changes to existing systems and processes, organizations can reduce their time used to maintain existing systems and free up IT people to focus on building new services that will help business to grow. Thus, SOA seems, and in fact proved to be, the correct strategy to meet the integration needs of the client.

Also, the use of the Oracle Retail suite represents a more than proven technology in the retail world. Thousands of retail and wholesale distribution companies around the world rely on Oracle for maximum flexibility and profitability. In fact, 20 of the top 20 global retailers run Oracle [Ora09a].

Regarding the Java CAPS framework, although being a restriction from DeSpar and also due to the already developed work with the framework, this can be considered as a good restriction, since it is experience of Wipro Retail, both with DeSpar, as with other customers, that the adoption of Java CAPS for the processes of EAI resulted in the following benefits:

- Simplification of the complexities of integration.
- Increase of business opportunity and value creation.
- Reduction of the time to market, by incrementing the IT flexibility and reactivity.
- Drastic reduction in IT costs, by pursuing reusable, template-based approaches to development.
- Make faster and better business decisions.
- Broad support for multiple computing platforms and databases.

Finally, to better understand how these technologies emerge in the project, Figure 3.15 describes the interconnection of the various technologies used throughout the internship.

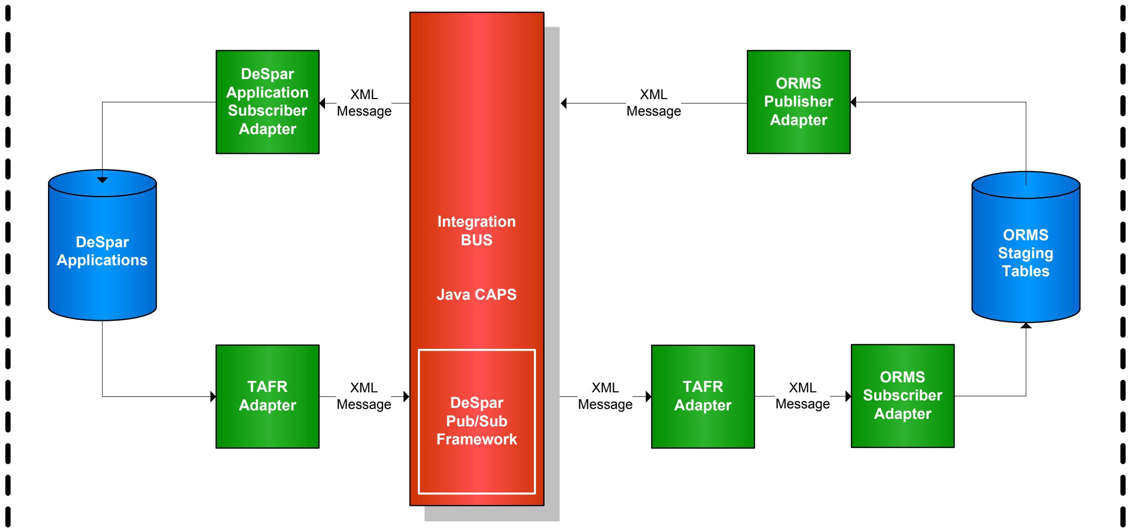


Figure 3.15: Project technologies overview

4 Enterprise Application Integration using Java CAPS

This section provides a detailed presentation of the problem that this project intends to solve, as well as the work developed throughout the internship. This work represents an integration solution, following an Enterprise Application Integration (EAI) approach and is described in the subsequent sub-sections.

4.1 Introduction

This internship is part of a bigger project from DeSpar, result of an Application Support and Maintenance (ASM) contract signed between DeSpar and Wipro Retail. Taking this into account, being part of an ASM project can be a much more challenging and enriching experience, as in the end proved to be, due to the possibility of dealing with various architectures and methodologies, accordingly to the client needs. Also, the fact of working in the EAI project team represents a great opportunity for learning, due to the technological variety involved in the client's support and change requests.

Thus, the work developed at Wipro Retail during the internship can be grouped into four major categories:

- **Store Productivity Application** – A productivity operational application, to store and manage data related to employees' efficiency in the stores.
- **Reception/RTV Change Request** – Follows a new data integration strategy regarding Receptions and Return-to-Vendor (RTV) data.
- **Store to Warehouse Orders and Receptions** – Consists in the development of an integration flow for data subscription regarding warehouse events for warehouse shipments and store-to-warehouse orders responses.
- **DeSpar Pub/Sub Framework Review** – Represents a needed review of the framework for versions merge and an impact analysing in existing projects.

The developed work is described in the following correspondent sub-sections.

4.2 Store Productivity Application

This application follows a DeSpar request for dismissing its current "Produttivitá" application, consisting in generic terms of a productivity operational application.

The project implements a Publisher/Subscriber model, for data publication by an AS400 system and subscription by the ORMS, regarding Store Productivity data. The AS400 system application will put files available to publish information and use the existing Publication/Subscription framework.

4.2.1 Assumptions

On the implementation of the Store Productivity integration flow should be taken into account the following assumptions:

- The EAI publication process will not include any data transformation.
- A staging area will be created for storing temporary data before committing it to the final ORMS tables. The created staging tables will hold processed data indefinitely. There will be a table field containing the processing date to allow the data purge. The staging tables purging process will be DeSpar responsibility.
- The data recovery will be made manually.
- The data will be referred as a message family, containing a message structure supported by a NB_EAI_<OBJECT> (Oracle Object Type). Any changes to the structure of the messages, will affect the specific message family and correspondent NB_EAI_<OBJECT>, the JCAPS OTD's and the Java Wrapper Classes.
- There will be a file for each store containing Store Productivity data. This file can be sent in different days with the same name. Every time the file is sent will be considered only the last data received as being the one to be processed into ORMS final tables.
- The file containing data will be available in a CSV format.
- The filename pattern will be "staff_[STORE]_[DATE].dat".

4.2.2 Architecture

The architecture of the Store Productivity integration flow to develop can be seen in Figure 4.1 that gives a general overview of components to implement, which are detailed in the subsequent sections.

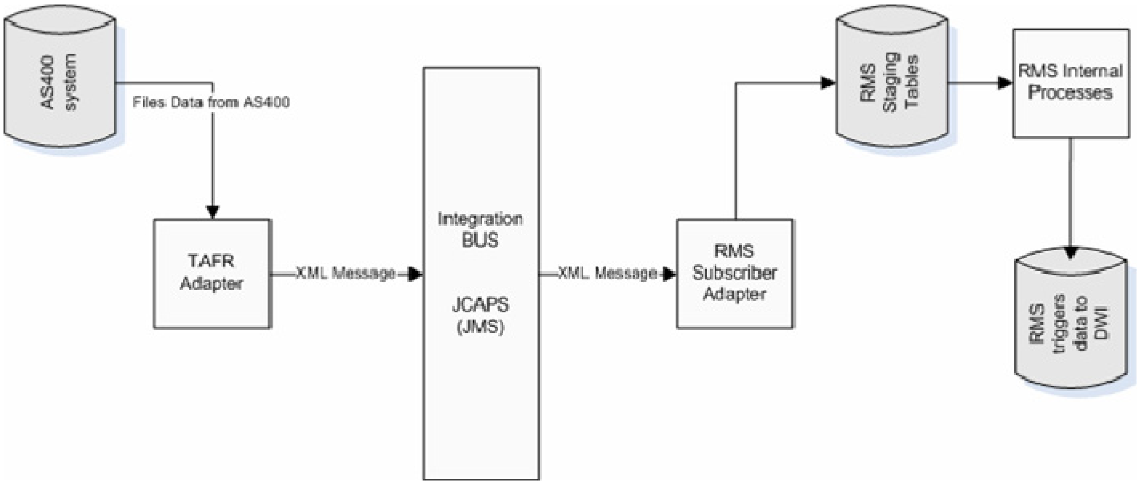


Figure 4.1: Store Productivity architecture

The Integration BUS will be used to receive data from external applications. This way AS400 system will send information about Store Productivity which will be integrated into ORMS using the mapping detailed in Table 4.1.

Table 4.1: Store Productivity mapping table

Publication		*Subscription*
Family	AS400 File Pattern	ORMS
StoreProductivity	staff_[store]_[date].dat	StoreProdDesc

4.2.3 GenericFromFTPToQ

This component will be used by the Store Productivity publisher, which will start by means of a triggering system that uses a JMS queue that holds a message responsible for initiating the publication business process. As its own name indicates, this component is a generic FTP component, therefore it may be used in any other JCAPS developments.

This developed component is responsible for getting files from a FTP host and send each file content to be processed using a Publisher/Subscriber approach. The logic implemented by the component is represented in Figure 4.2.

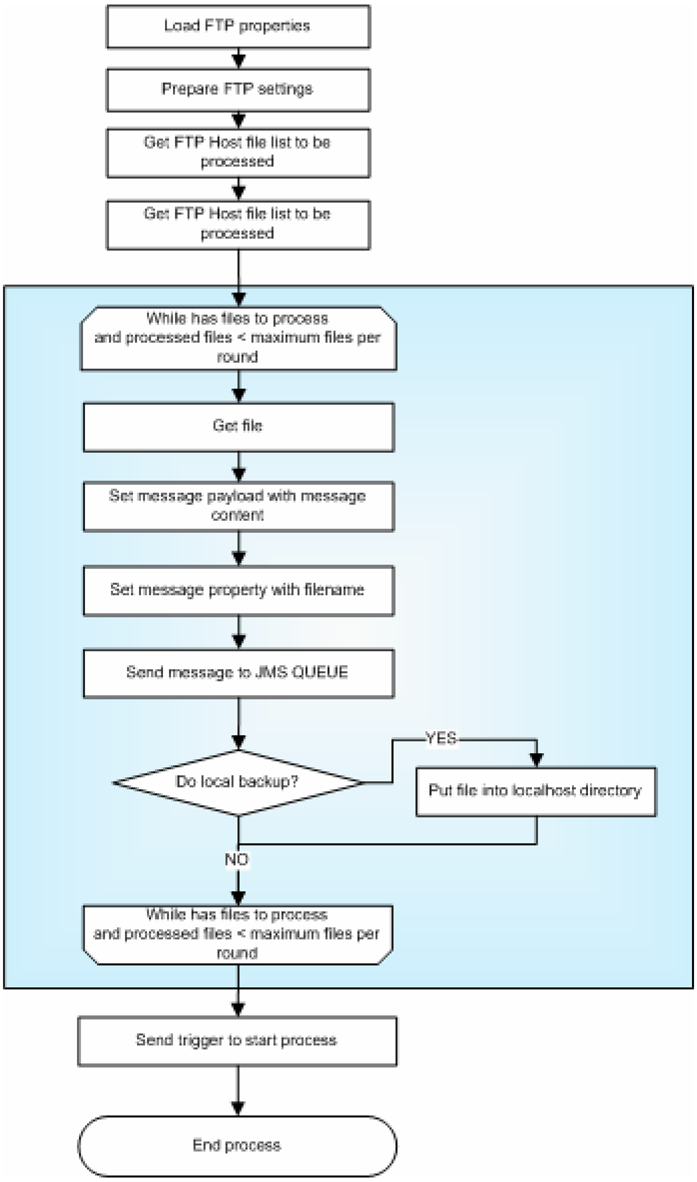

Load FTP properties
Prepare FTP settings
Get FTP Host file list to be processed
Get FTP Host file list to be processed
While has files to process and processed files < maximum files per round
Get file
Set message payload with message content
Set message property with filename
Send message to JMS QUEUE
Do local backup?
YES
Put file into localhost directory
NO
While has files to process and processed files < maximum files per round
Send trigger to start process
End process

Figure 4.2: GenericFromFTPToQ component logic

The FTP access to the AS400 system, which will retrieve files with the data to be sent to the integration BUS, is parameterized by a configuration file described in Table 4.2.

Table 4.2: GenericFromFTPToQ configuration

GenericFromFTPToQ			
Object	Type	New?	Description
genericFromFTPToQ.properties	Properties file	Y	Configuration file for the FTP access to the AS400 system, which includes the following new entries: • maxFilesPerRound = the number of files to be processed in each FTP. • hostDirectory = the folder name in the FTP host to retrieve the files. • hostFileName = the filename pattern to retrieve. • hostPostCommand = the command to execute after retrieve the file. • hostPostDirectory = the name of the directory in which to put the processed files. • hostPostFileName = the filename pattern for the processed files. • localBackupDirectory = the folder name in which to backup files locally (in JCAPS). • trigger.host = the name of the host where the trigger signal is to be sent • trigger.port = the port number of the queue in the host which will receive the trigger signal. • trigger.queue = the name of the internal queue to send the trigger signal.

As for the FTP component logic, this was implemented in JCAPS using a Java Collaboration Definition (JCD). This collaboration, *svcGenericFromFTPToQ*, was then linked with other logical components needed, by using a connectivity map, as illustrated in Figure 4.3.

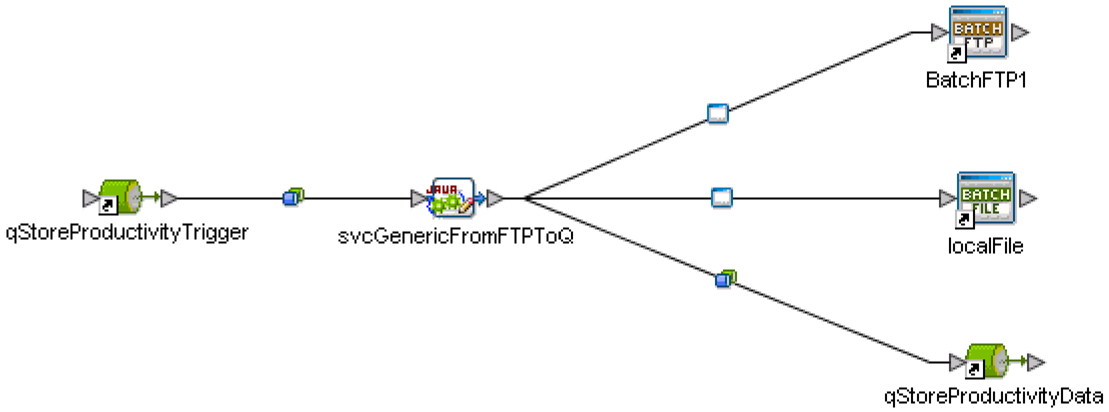


Figure 4.3: GenericFromFTPToQ connectivity map definition

The components used in the connectivity map and illustrated above, are detailed in Table 4.3.

Table 4.3: GenericFromFTPToQ components

GenericFromFTPToQ components			
Object	Type	New?	Description
qStoreProductivityTrigger	JMS queue	Y	JMS queue that holds the message to start the publication business process.
svcGenericFromFTPToQ	Java Collaboration	Y	It is responsible for the FTP access to the AS400 system, which will retrieve files with the data to be sent to the integration BUS.
BatchFTP1	FTP adapter	Y	Provides FTP access.
localFile	File adapter	Y	Allows access to the FTP properties file.
qStoreProductivityData	JMS queue	Y	JMS queue that holds the Store Productivity data retrieved from the CSV files.

4.2.4 Store Productivity Publisher

This component is responsible for the data publication of the Store Productivity messages into the integration BUS. The data publication process will start by means of a triggering system, which uses a JMS queue that holds the message to initiate the publication business process.

Thus, in general terms, the publication flow will start by executing a FTP access to the AS400 system, which will retrieve the CSV format files with the data to be sent to the integration BUS, and then will map the file content to a XML message structure using an Object Type Definition (OTD). An OTD is a JCAPS component used to define and manipulate message structures. To allow other components to manipulate fields within JCAPS messages, the OTD provides both marshal (serialize) and unmarshal (deserialize) methods.

The OTD, *StoreProductivity*, used to map the received CSV files, via FTP, to XML format, is detailed in Appendix A.1.

Since the goal is to insert the retrieved CSV files data into the staging area tables, it is necessary to map the flat file format against the database format, which is achieved by defining a Common Message Format (CMF) specification that basically defines the structure of the messages, shared by publishers and subscribers, of the integration flow. The CMF definition has been accomplished by using another OTD, which maps the file contents already in XML format to the same structure defined in the Oracle Object Types, user-defined data types that will model the Store Productivity data as unitary entities (objects) to allow the integration in the ORMS.

At this point, the OTD has been defined by means of a Document Type Definition (DTD), and then imported to JCAPS, that automatically generates an OTD with the given DTD. The structure of the generated OTD, *StoreProdDesc*, is described in Appendix B.1.

So, now the integration BUS contains the data formatted and ready to be sent for publication, which is achieved by sending the message to a JMS topic, thus concluding the publication flow. All the logic developed for the publication has been implemented using a collaboration, *jcdStoreProductivityPublisher* (Appendix F.1), and included in the connectivity map represented in Figure 4.4.

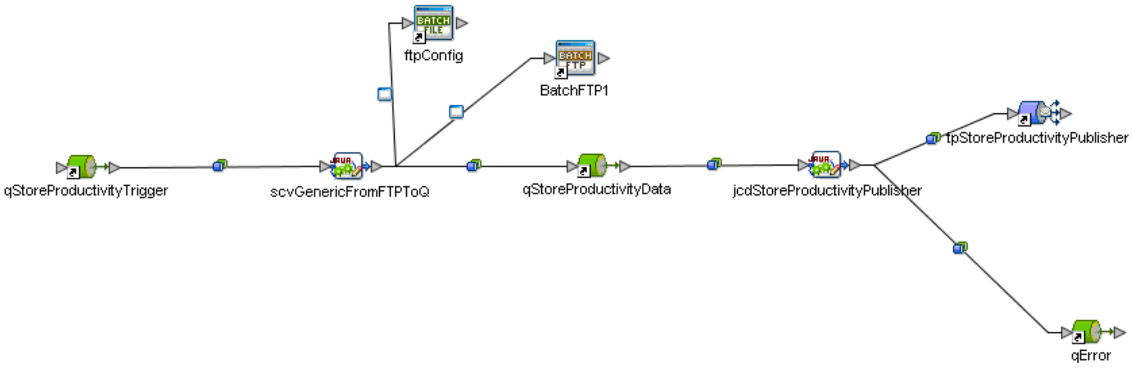

ftpConfig
BatchFTP1
tpStoreProductivityPublisher
qStoreProductivityTrigger
scvGenericFromFTPToQ
qStoreProductivityData
jcdStoreProductivityPublisher
qError

Figure 4.4: Store Productivity publisher connectivity map definition

Table 4.4 describes the components used in the Store Productivity publisher connectivity map.

Table 4.4: Store Productivity publisher components

Store Productivity publication components			
Object	Type	New?	Description
qStoreProductivityTrigger	JMS queue	Y	JMS queue that holds the message to start the publication business process.
svcGenericFromFTPToQ	Java Collaboration	Y	It is responsible for the FTP access to the AS400 system, which will retrieve files with the data to be sent to the integration BUS.
ftpConfig	File adapter	Y	Allows access to the FTP properties file.
BatchFTP1	FTP adapter	Y	Provides FTP access.
qStoreProductivityData	JMS queue	Y	JMS queue that holds the Store Productivity data retrieved from the CSV files.
jcdStoreProductivityPublisher	JMS queue	Y	It is responsible to receive and format the data, from the integration Bus, to be ready for publication.
tpStoreProductivityPublisher	JMS topic	Y	Contains the data formatted from the integration BUS.
qError	JMS queue	Y	JMS queue that holds any eventual error message that occurred during the publication process.

4.2.5 ORMS Subscriber

For the ORMS subscriber for Store Productivity data it is necessary to create the Java Wrapper Classes, in order to make possible for the framework to interchange data between systems, in this case between JCAPS and the ORMS.

The Java Wrapper Classes are responsible for the Oracle Objects initialization, taking advantage of the Java Reflection API to read and write fields and methods of a selected class in runtime, namely the methods used to map the CMF messages into the Oracle Object Types.

Thus, the data formatted according to the OTD, *StoreProdDesc* (Appendix B.1), is received from the integration BUS and mapped via Java Wrapper Classes into the developed Oracle Object Types. After creating the Oracle Object Types, the Java Wrapper Classes are easily generated using Oracle JDeveloper, a free IDE from Oracle, which simplifies the development of Java-based SOA applications and user interfaces with support for the full development life cycle.

Regarding the Store Productivity subscriber, a connectivity map has been defined, with a collaboration, *jcdStoreProductivityToRMS* (Appendix F.5), to manipulate the received messages and connect to the ORMS, in order to consume the data and integrate it with the destination staging tables. Figure 4.5 illustrates the configuration of the developed subscription flow.

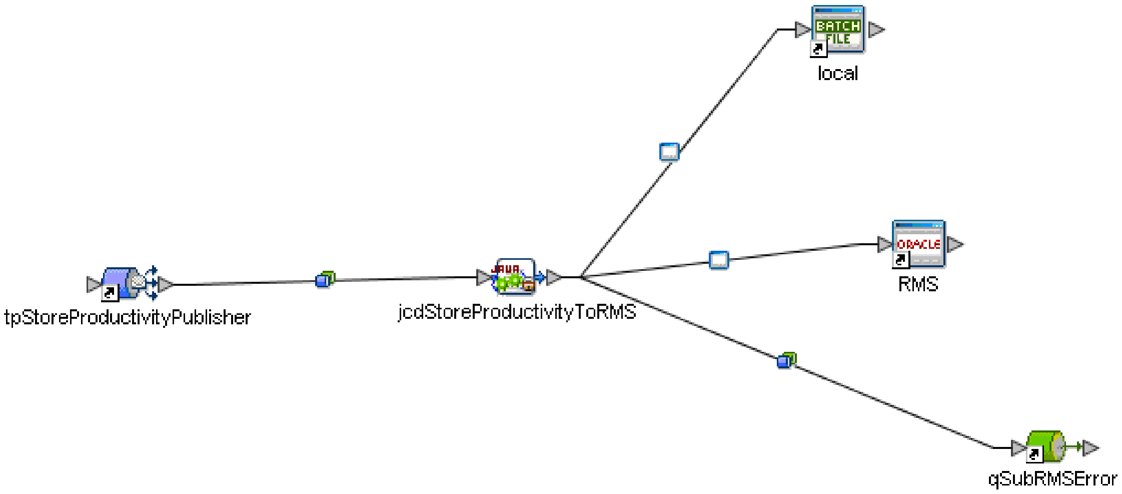


Figure 4.5: Store Productivity ORMS subscriber connectivity map definition

The components represented in the figure above are described in Table 4.5.

Table 4.5: Store Productivity ORMS subscriber components

Store Productivity subscription components			
Object	Type	New?	Description
tpStoreProductivityPublisher	JMS topic	Y	Contains the data formatted from the integration BUS.
jcdStoreProductivityToRMS	Java Collaboration	Y	It is responsible to put the XML message from the integration Bus into the ORMS staging tables.
local	File adapter	Y	Allows access to the subscription properties file.
RMS	Oracle External Application	Y	Provides an Oracle database connection.
qSubRMSError	JMS queue	Y	JMS queue that holds any eventual error message that occurred during the subscription process.

4.2.6 Data Model

As previously described, an assumption for the implementation of the Store Productivity application was the creation of a staging area for storing temporary data before committing it to the final ORMS tables.

The staging tables are designed based on the message specification and the necessities of the ORMS. Through the combination of these two factors, and analysing the message specification detailed in Appendix B.1, the typical approach used for the client, in this and other projects, consists of defining a Common Message Format (CMF) that is divided in the two following parts:

- **StoreProdDesc** – It is composed by a header which contains the standard features of the entity, in this case the Store Productivity data description.
- **StoreProdDtlDesc** – This other part is formed by a logic of detail, to contain the employees' efficiency data in the store associated with the header of the message.

The CMF message is mapped via Java Wrapper Classes into the Oracle Object Types and then the data is inserted on the staging tables with the aid of a developed subscription package. Basically, to each of those Store Productivity CMF parts mentioned above correspond a staging table, as described bellow:

- **NB_EAI_STORE_PROD_STG** – Staging table that will hold the Store Productivity data description correspondent to the *StoreProdDesc* part of the CMF.
- **NB_EAI_STORE_PROD_DTL_STG** – On this table will be inserted the employees' efficiency data contained in the *StoreProdDtlDesc* CMF element.

More information on the data model used in the Store Productivity application, namely the staging tables description and message specification, are detailed in Appendixes A to C.

4.2.7 Subscription Package

The subscription package is an integration component which is part of the methodology of database sharing. Since many applications share the same database schema it is necessary to encapsulate all common database interactions, the so-called CRUD (Create, Read, Update and Delete) operations. This integration mechanism is accomplished with the development of packages that will insert the data mapped in the Oracle Object Type into the database staging tables.

Thus, a subscription package represents a reusable component, by applications that use the same logic of data integration, and allow for a fast and efficient execution of the CRUD operations directly to the database, since the whole process is managed by the DBMS (Database Management System).

The developed package uses the NB_RMSSUB_<BUSINESS_ENTITY> naming convention, in this case NB_RMSSUB_STORE_PROD, since it is a Store Productivity application. The package procedures and functions are described bellow:

- **Procedure CONSUME** – This is the main procedure of the package, which receives the Oracle Object sent by JCAPS and manages its integration logic into the ORMS.
- **Function PROCESS_STAGING** – This function will be launched by the CONSUME procedure and will load into the ORMS staging tables the information inside the Oracle Object.
- **Function PROCESS_CORE** – This function will be launched by the CONSUME function, after the PROCESS_STAGING function has finished and will insert the temporary data in the staging area into the final ORMS tables, if no error occurred during the staging phase.

The CONSUME algorithm implemented by the subscription package is represented by the flowchart in Figure 4.6.

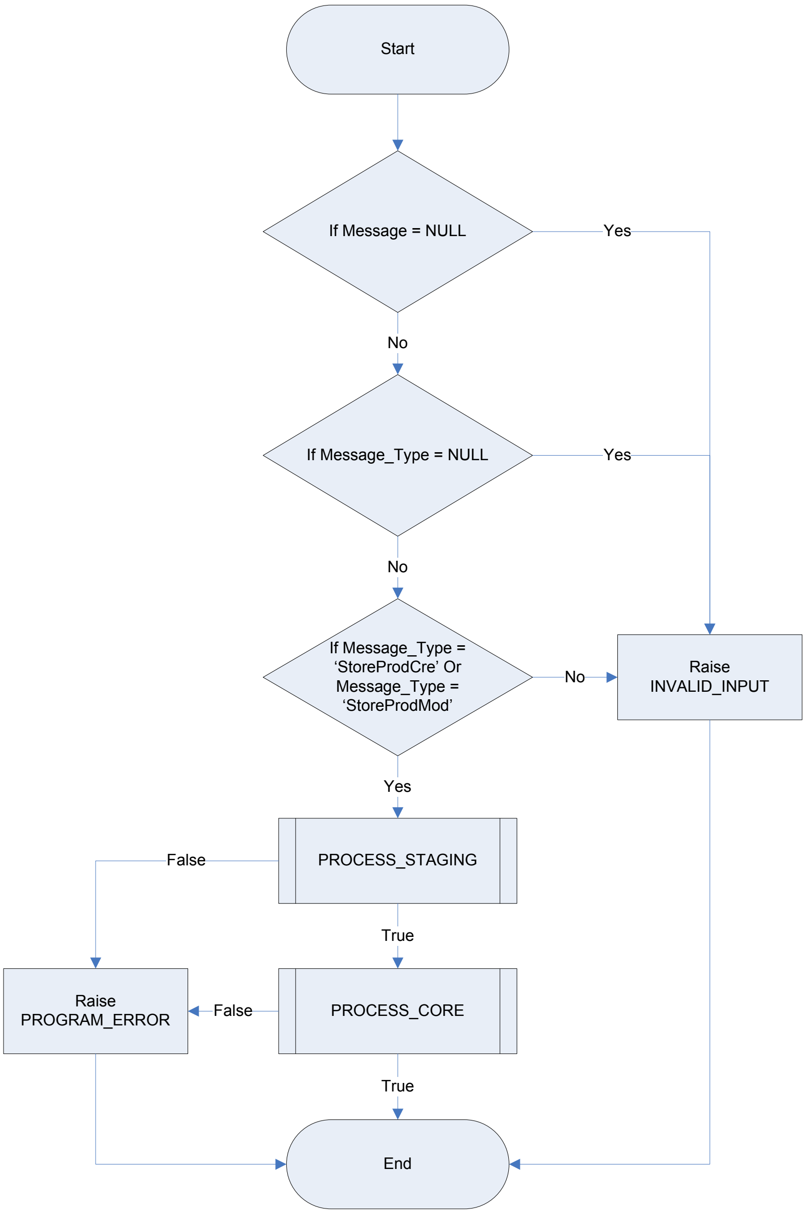


Figure 4.6: Store Productivity CONSUME algorithm

A more detailed description of the NB_RMSSUB_STORE_PROD package, namely its procedures and functions, can be found in Appendix D.1.

4.3 Reception/RTV Change Request

This sub-section describes another development made in the context of the EAI strategy being implemented for DeSpar.

The Reception/RTV consists of a change request by DeSpar that aims for the implementation of an integration flow between the IMAge and the ORMS, and comprises the two following business entities:

- **Reception** – This entity includes all the data relating to orders receipt by the retailer.
- **Return-to-Vendor (RTV)** – Similar to the Receptions, the RTV entity also deals with items but with the difference that these need to be returned to the vendor.

The link between these two entities, into the same integration flow, is due to the fact that they share the same message structure which is received from the IMAge and sent into the integration BUS.

In resemblance with the Store Productivity application, this change request implements a Publisher/Subscriber model, for data publication by the IMAge system and subscription by the ORMS, regarding Receptions and RTV data.

One version of the integration flow for the Common Message Format (CMF) referent to the Receptions is already implemented, although some changes are required in order for the two entities (Receptions and RTV) to be integrated and work as one. As for the RTV, a new implementation will be made from scratch.

4.3.1 Assumptions

On the development of the Reception/RTV integration flow should be taken into account the following assumptions:

- The EAI publication process will not include any data transformation.
- A staging area will be created for storing temporary data before committing it to the final ORMS tables. The created staging tables will hold processed data indefinitely. There will be a table field containing the processing date to allow the data purge. The staging tables purging process will be DeSpar responsibility.
- The data recovery will be made manually.
- Each message family will use a different NB_EAI_<OBJECT> (Oracle Object Type). Any changes to the structure of the messages, will affect the specific message family and correspondent NB_EAI_<OBJECT>, the JCAPS OTD's and the Java Wrapper Classes.
- The external information block concept (EXT_INFO) will be only used as an exception measure.

4.3.2 Architecture

The architecture of the Reception/RTV integration flow to develop can be seen in Figure 4.7 that gives a general overview of components to implement, which are detailed in the subsequent sections.

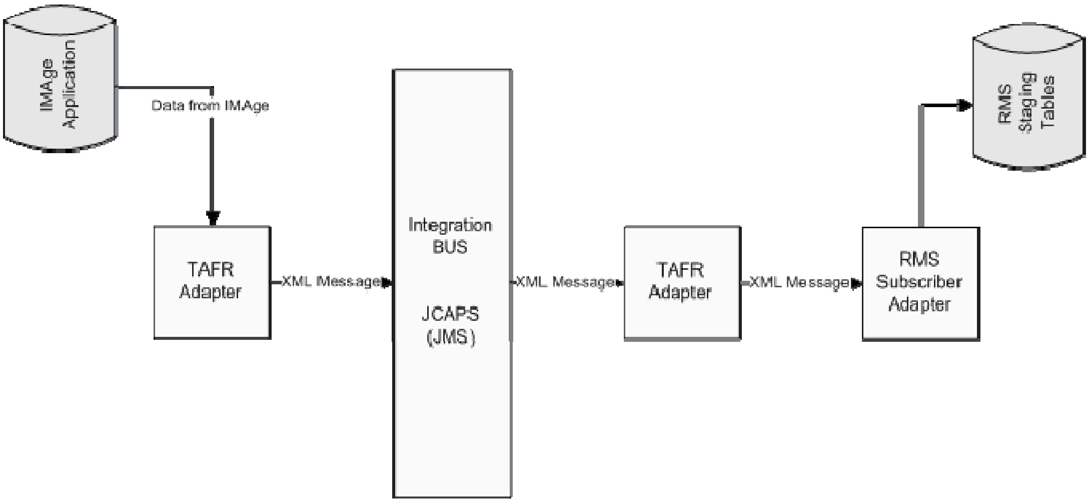


Figure 4.7: Reception/RTV architecture

The integration BUS will be used to receive data from external applications. This way IMAge application will send information about Receptions/RTV which will be integrated into ORMS using the mapping detailed in Table 4.6.

Table 4.6: Reception/RTV mapping table

Publication		*Subscription*
Family	IMAge data	ORMS
Reception	FWZ1	ReceptionDesc
	FWZ2	
RTV	FWZ1	RTVDesc
	FWZ2	

4.3.3 IMAge Publishers

4.3.3.1 ImageOutputGateway

This component is responsible for all IMAge publications into the integration BUS. It executes a specific IMAge SQL query, which retrieves the data to be sent to the integration BUS.

This component has been updated to deal with the new RTV message types, namely in its configuration properties which are described in Table 4.7.

Table 4.7: ImageOutputGateway configuration

ImageOutputGateway			
Object	Type	New?	Description
Publication_FromImage .properties	Properties file	N	Configurations file containing the mapping between the message type received and the destination JMS queue to send the data. This file will include the new entries: • FWZ.Host = <JCAPS_host> • FWZ.Port = <PORT_NUMBER> • FWZ.Type = Q • FWZ.Dest = qImageTAFRInput • TAFR.Reception.Host = <JCAPS_host> • TAFR.Reception.Port = <PORT_NUMBER> • TAFR.Reception.Type = Q • TAFR.Reception.Dest = qReceptionInput • TAFR.RTV.Host = <JCAPS_host> • TAFR.RTV.Port = <PORT_NUMBER> • TAFR.RTV.Type = Q • TAFR.RTV.Dest = qRTVInput

As the Reception and RTV messages are received with the same IMAge message format, all of the received messages with the FWZ specific format are sent to its specific JMS queue, to be analyzed and then sent to the configured destination.

4.3.3.2 FromImageTAFR

The objective of the development of this component is to deal with specific Transformation, Address, Filtering and Routing (TAFR) specifications with the received messages in the JMS queue, and will be used to select the appropriate JMS destination to deal with the received message (qReceptionInput/qRTVInput).

In the current interface, it is only used the routing capability of this TAFR component to route messages based on the FWZ1.PO_TYPE value from the IMAge message received. The PO_TYPE field designates the type of purchase order, thereby allowing to identify the kind of order the message refers as it follows:

- PO_TYPE value is different of '99' will be considered as a Reception
- PO_TYPE value is equal to '99' will be considered as a RTV

The developed TAFR has been implemented using a collaboration, *svcFromImageTAFR* (Appendix F.2), and follows the configuration represented in Figure 4.8.

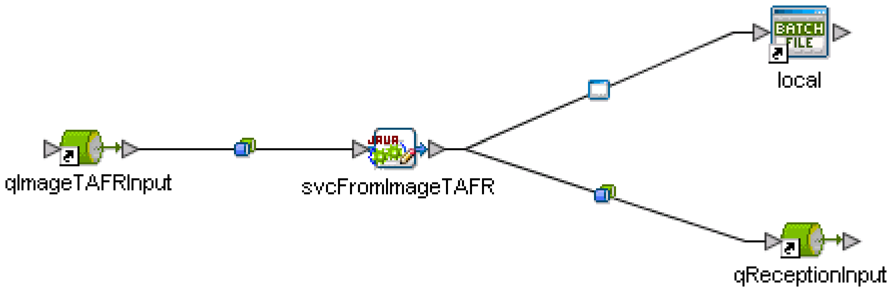


Figure 4.8: FromImageTAFR connectivity map definition

Table 4.8 describes the components used in the FromImageTAFR connectivity map.

Table 4.8: FromImageTAFR components

FromImageTAFR components			
Object	Type	New?	Description
qImageTAFRInput	JMS queue	Y	The JMS queue which will have the IMAge messages to deal with.
svcFromImageTAFR	Java Collaboration	Y	It is responsible for identifying the entity type (Reception or RTV) and send it to the correspondent destination queue.
local	File adapter	Y	Allows access to the TAFR properties file.
qReceptionInput	JMS queue	Y	The destination queue which holds the Reception/RTV messages to be sent to the correspondent publisher.

4.3.3.3 Reception Publisher

The Reception publisher is responsible for the data publication of the Reception messages into the integration BUS.

In order to map the IMAge messages, an OTD, *ReceptionDesc*, has been created, to have the desired message structure defined, to send it for subscription. This OTD has been generated via a DTD, and is detailed in Appendix B.2.

All the logic developed for the Reception data publication has been implemented using a collaboration, *jcdReceptionGetFromImage* (Appendix F.3), which has been included in the connectivity map illustrated in Figure 4.9.

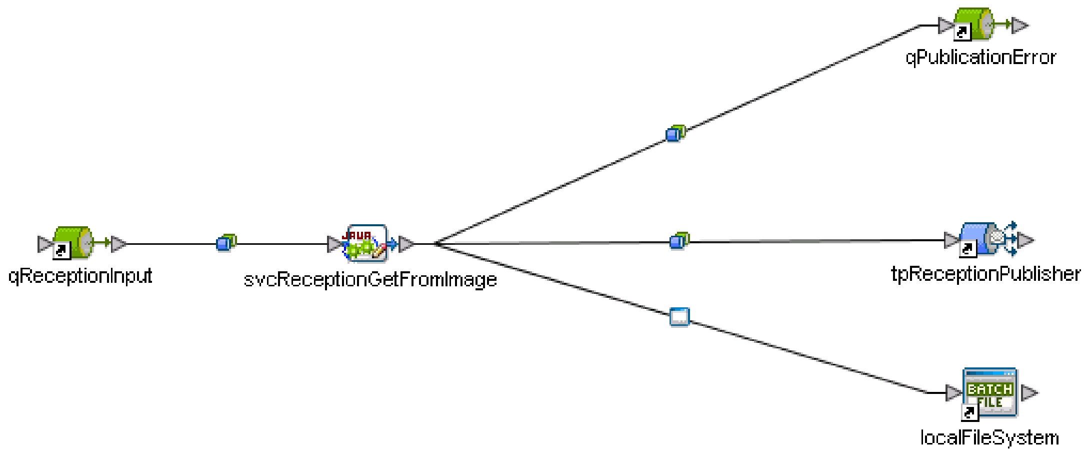


Figure 4.9: Reception publisher connectivity map definition

The components represented in the figure above are described in Table 4.9.

Table 4.9: Reception publisher components

Reception publication components			
Object	Type	New?	Description
qReceptionInput	JMS queue	N	Contains the Receptions retrieved from IMAge (after XML mapping).
svcReceptionGetFromImage	Java Collaboration	N	It gets the Receptions message from the integration Bus, maps and formats it into XML and puts it back in the integration Bus to be processed using the Pub/Sub approach.
qPublicationError	JMS queue	N	JMS queue that holds any eventual error message that occurred during the publication process.
tpReceptionPublisher	JMS topic	N	Contains the data formatted from the integration BUS.
localFileSystem	File adapter	N	Allows access to the publication properties file.

4.3.3.4 RTV Publisher

This component is responsible for the data publication of the RTV messages into the integration BUS.

In order to map the IMAge messages, an OTD, *RTVDesc*, has been created, to have the desired message structure defined, to send it for subscription. This OTD has been generated via a DTD, and is detailed in Appendix B.3.

All the logic developed for the RTV data publication has been implemented using a collaboration, *jcdRTVGetFromImage* (Appendix F.4), which has been included in the connectivity map illustrated in Figure 4.10.

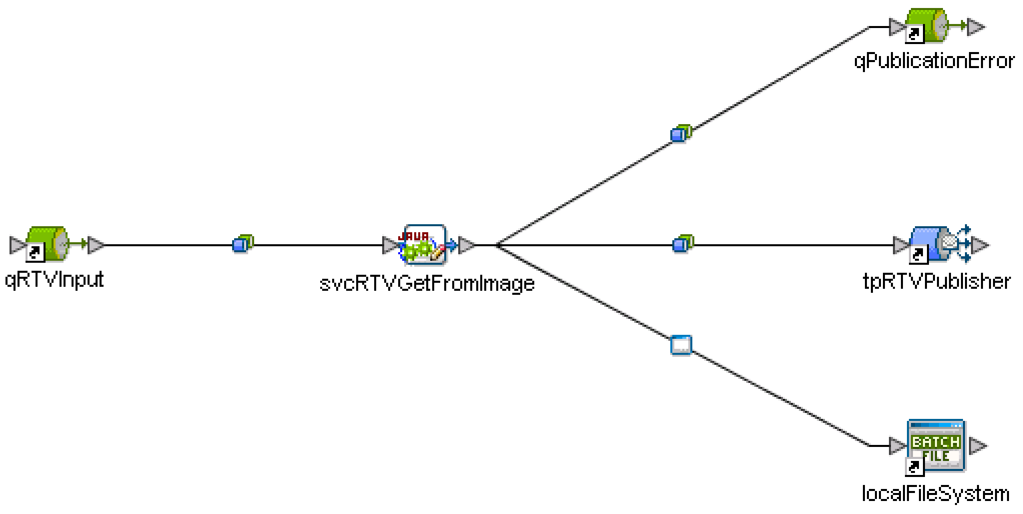


Figure 4.10: RTV publisher connectivity map definition

The components used in the connectivity map and illustrated above, are detailed in Table 4.10.

Table 4.10: RTV publisher components

RTV publication components			
Object	Type	New?	Description
qRTVInput	JMS queue	Y	Contains the RTV's retrieved from IMAge (after XML mapping).
svcRTVGetFromImage	Java Collaboration	Y	It gets the RTV message from the integration Bus, maps and formats it into XML and puts it back in the integration Bus to be processed using the Pub/Sub approach.
qPublicationError	JMS queue	Y	JMS queue that holds any eventual error message that occurred during the publication process.
tpRTVPublisher	JMS topic	Y	Contains the data formatted from the integration BUS.
localFileSystem	File adapter	Y	Allows access to the publication properties file.

4.3.4 ORMS Subscribers

4.3.4.1 Reception Subscriber

In relation to the Reception data subscription, the necessary Java Wrapper Classes have been created and added to the JCAPS project, in order to make possible for the framework to interchange data between systems, in this case between JCAPS and the ORMS. Therefore, the

Java Wrapper Classes are responsible for translating the data, previously formatted according to the OTD, *ReceptionDesc* (Appendix B.2), and received from the integration BUS, into the developed Oracle Object Types. After creating the Oracle Object Types, the Java Wrapper Classes are easily generated using Oracle JDeveloper.

For the ORMS subscriber for IMAge Reception data, it will pick the data from the integration BUS and insert it into the ORMS staging tables.

The subscriber is implemented by a collaboration, *svcReceptionToRMS* (Appendix F.5), and uses the configuration described in Figure 4.11.

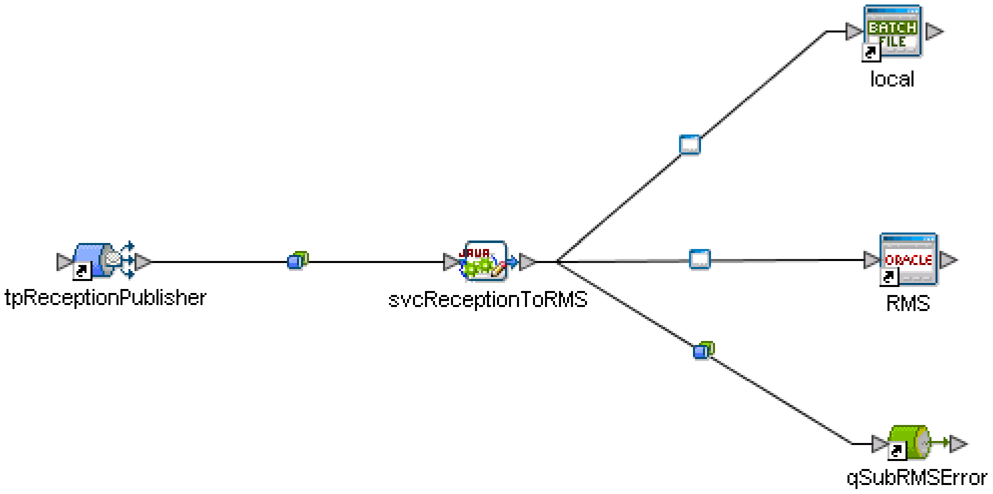


Figure 4.11: Reception ORMS subscriber connectivity map definition

The components used in the connectivity map and illustrated above, are detailed in Table 4.11.

Table 4.11: Reception ORMS subscriber components

Reception subscription components			
Object	Type	New?	Description
tpReceptionPublisher	JMS topic	N	Contains the data formatted from the integration BUS.
svcReceptionToRMS	Java Collaboration	N	It is responsible to put the XML message from the integration Bus into the ORMS staging tables.
Local	File adapter	N	Allows access to the subscription properties file.
RMS	Oracle External Application	N	Provides an Oracle database connection.
qSubRMSError	JMS queue	N	JMS queue that holds any eventual error message that occurred during the subscription process.

4.3.4.2 RTV Subscriber

For the ORMS subscriber for RTV data the necessary Java Wrapper Classes have been created and added to the JCAPS project, similarly to the Reception publisher, in order to make possible for the framework to interchange data between systems, in this case between JCAPS and the ORMS. Thus, the data formatted according to the OTD, *RTVDesc* (Appendix B.3), is received from the integration BUS and mapped via Java Wrapper Classes into the developed Oracle Object Types. As for other projects, the Java Wrapper Classes are generated using Oracle JDeveloper.

The objective of the ORMS subscriber for IMAge RTV data is to pick the data from the integration BUS and insert it into the ORMS staging tables.

The subscriber is implemented by a collaboration, *svcRTVToRMS* (Appendix F.5), and uses the configuration described in Figure 4.12.

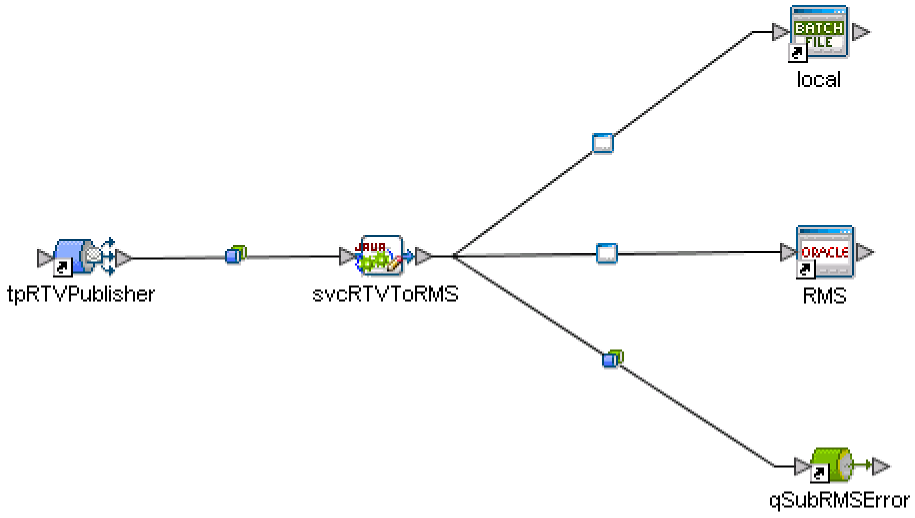


Figure 4.12: RTV ORMS subscriber connectivity map definition

The components represented in the figure above are described in Table 4.12.

Table 4.12: RTV ORMS subscriber components

RTV subscription components			
Object	Type	New?	Description
tpRTVPublisher	JMS topic	Y	Contains the data formatted from the integration BUS.
svcRTVToRMS	Java Collaboration	Y	It is responsible to put the XML message from the integration Bus into the ORMS staging tables.
local	File adapter	Y	Allows access to the subscription properties file.
RMS	Oracle External Application	Y	Provides an Oracle database connection.
qSubRMSError	JMS queue	Y	JMS queue that holds any eventual error message that occurred during the subscription process.

4.3.5 Data Model

As previously described, the two business entities involved in this project, Reception and RTV, share the same IMAge message specification, which in this case corresponds to the FWZ message format.

The FWZ definition is divided in the following two parts:

- **FWZ1** – It is composed by the IMAge message header.
- **FWZ2** – Formed by a logic of detail, which contains the IMAge message details associated with the header of the message.

Tables 4.13 and 4.14 describe which fields of the FWZ IMAge message are mapped into a Reception or RTV message structure.

Table 4.13: FWZ1 to Reception/RTV mapping table

FWZ1	*Reception*	*RTV*
LOCATION	√	√
ACQUISITION_NR	√	√
TRANSACTION_NUMBER	√	√
TRANSACTION_DATE	√	√
PLAN_ACQUISITION_DATE	√	√
PLAN_ACQUISITION_SYSDATE	√	√
SUPPLIER	√	√
RET_AUTH_NUM	√	√
ARRIVAL_DATE	√	√
SHIPMENT_DATE	√	√
USER_CODE	√	√
PO_TYPE	√	√
UPC_DESC	√	
BOOKDATE	√	√
ACQUISITION_NR_REF	√	
RET_AUTH_NUM_REF	√	
ARRIVAL_DATE_REF	√	
UPC	√	√

Table 4.14: FWZ2 to Reception/RTV mapping table

FWZ2	*Reception*	*RTV*
LOCATION	√	
ACQUISITION_NR	√	
TRANSACTION_NUMBER	√	
TRANSACTION_SEQ_NO	√	√
SKU	√	√
ACQUISITION_SKU	√	√
WH	√	√
SUPP_PACK_SIZE_QTY	√	√
SUPP_PACK_SIZE	√	√
QTY_RETURNED	√	√
DISCOUNT_TYPE	√	
UNIT_COST_SUP_BASE	√	
DISCOUNT_RATE_1	√	
DISCOUNT_RATE_2	√	
UNIT_COST_ACTUAL	√	
UNIT_COST	√	√
CHAINING_BASE	√	√
CHAINING_FACTOR	√	√
STATUS	√	√
BUY_PRICE	√	√
AUTO_SPK_QTY	√	√

AUTO_RETURN_QTY	√	√
ITEM_TYPE	√	√
RESERVATION_LOGIC	√	√
UNIT_COST_DIRECT	√	√
UNIT_COST_CONTROL	√	
BOL_POSITION_NR	√	

As previously described, an assumption for the implementation of the Reception/RTV change request was the creation of a staging area for storing temporary data before committing it to the final ORMS tables.

The Reception and RTV staging tables already exist in the ORMS, however a few changes were required to some fields of these tables due to changes that occurred in the IMAge messages definition.

The CMF for the Reception/RTV project is divided in the two following parts:

- **ReceptionDesc** – It is composed by a header which contains the standard features of the entity, in this case Reception data description.
- **ReceptionDetailDesc** – This other part is formed by a logic of detail, to contain the Receptions' features data associated with the header of the message.
- **RTVDesc** – This represents the standard features of the RTV, which correspond to the header of the message.
- **RTVDetailDesc** – Corresponds to the logic of detail relative to the RTV's, which is associated with the *RTVDesc*.

The Reception/RTV CMF message is mapped via Java Wrapper Classes into the Oracle Object Types and then the data is inserted on the staging tables with the aid of a developed subscription package. Basically, to each of those Reception/RTV CMF parts mentioned above correspond a staging table, as described bellow:

- **NB_EAI_RECEPTION_STG** – Staging table that will hold the Reception data description correspondent to the *ReceptionDesc* part of the CMF.
- **NB_EAI_RECEPTION_DETAIL_STG** – On this table will be inserted the Receptions' features data contained in the *ReceptionDetailDesc* CMF element.
- **NB_EAI_RTV_STG** – This is the RTV staging table where will be inserted the RTV data description concerning the *ReceptionDesc* part of the CMF.
- **NB_EAI_RTV_DETAIL_STG** – On this table will be inserted the RTV's features data contained in the *RTVDetailDesc* CMF element.

More information on the data model used in the Reception/RTV change request, namely the staging tables description and message specification, are detailed in Appendixes A to C.

4.3.6 Subscription Packages

These components deal with the data integration into the ORMS staging tables, following a database sharing methodology, as in the Store Productivity application.

The usage of subscription packages encapsulates all common database interactions, the CRUD (Create, Read, Update and Delete) operations, resulting in a reusable component that will insert the data mapped in the Oracle Object Type into the database staging tables.

The subscription packages naming follows the usual practice adopted in this kind of EAI projects which is the NB_RMSSUB_<BUSINESS_ENTITY> naming convention. In this particular case, two packages have been developed – NB_RMSSUB_RECEPTION and NB_RMSSUB_RTV – corresponding to the Reception and RTV business entities.

The packages procedures and functions are described bellow:

- **Procedure CONSUME** – This is the main procedure of the package, which receives the Oracle Object sent by JCAPS and manages its integration logic into the ORMS.

- **Function PROCESS_STAGING** – This function will be launched by the CONSUME procedure and will load into the ORMS staging tables the information inside the Oracle Object.
- **Function PROCESS_CORE** – This function will be launched by the CONSUME function, after the PROCESS_STAGING function has finished and will insert the temporary data in the staging area into the final ORMS tables, if no error occurred during the staging phase.

The CONSUME algorithms implemented for both subscription packages are very similar and are represented by the flowcharts in Figures 4.13 and 4.14.

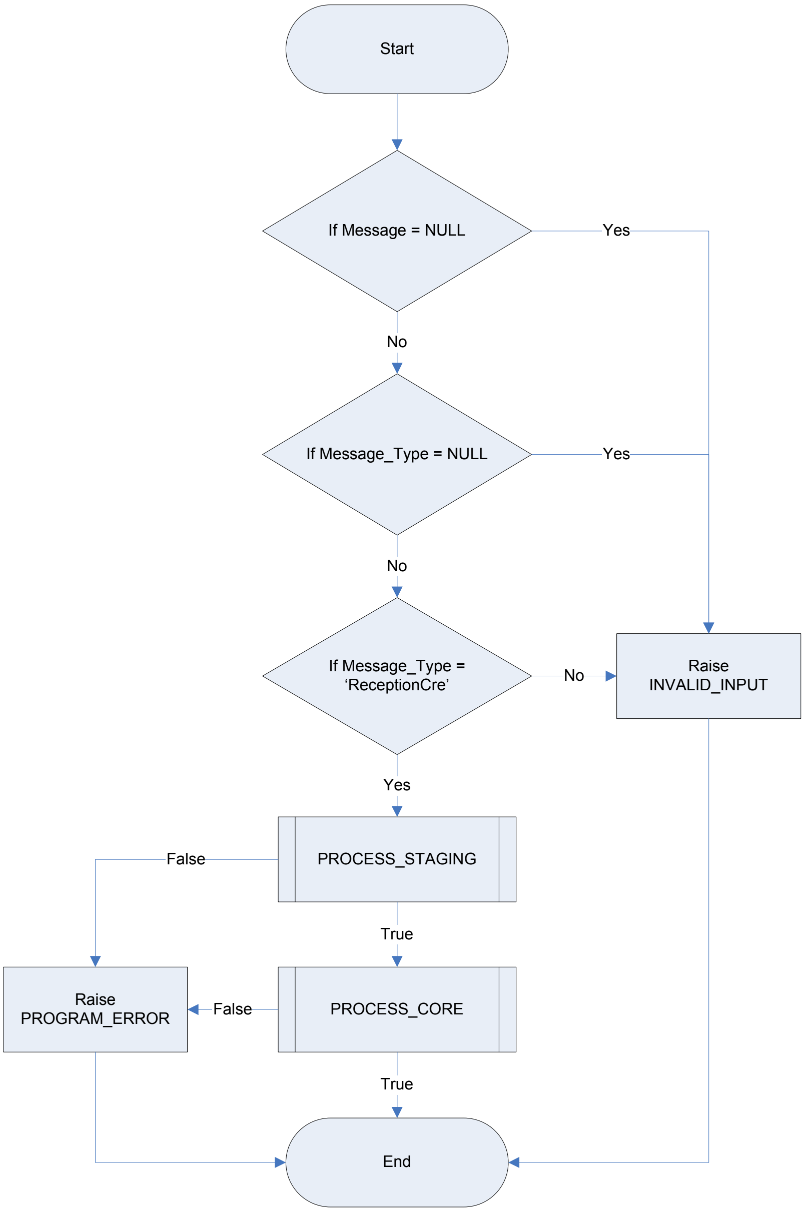


Figure 4.13: Reception CONSUME algorithm

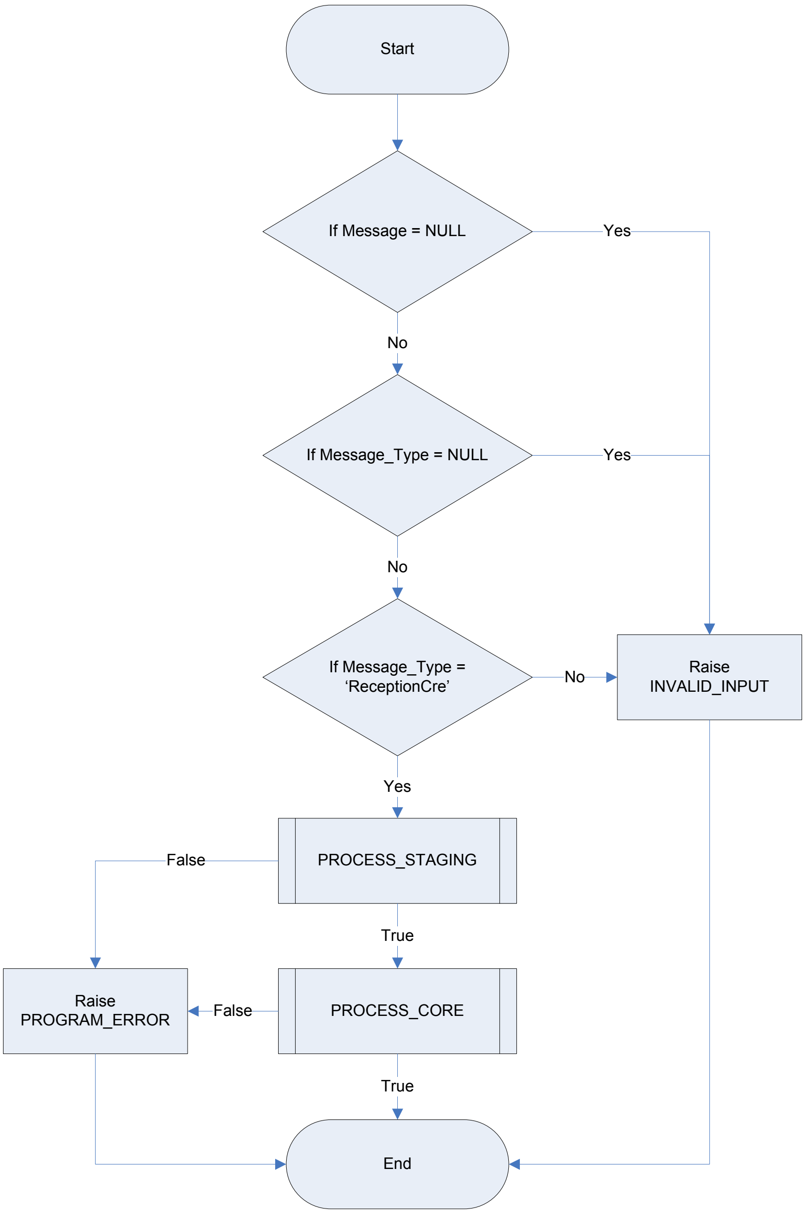


Figure 4.14: RTV CONSUME algorithm

A more detailed description of the NB_RMSSUB_RECEPTION and NB_RMSSUB_RTV packages, namely its procedures and functions, can be found in Appendixes D.2 and D.3, correspondingly.

4.4 Store to Warehouse Orders and Receptions

Store to Warehouse Orders and Receptions is the designation given to another DeSpar request for data integration, in this specific case from the ORMS application publishers to the integration BUS and correspondent IMAge data subscription, regarding warehouse events, namely warehouse shipments and store-to-warehouse orders responses.

Since this project deals with warehouse notifications, a business entity named WhNotification will be used, which comprises the two following entities:

- **Shipment** – This entity contains information, which is sent by the ORMS application to the IMAge, concerning shipped items, within a given store order.

- **OrderResponse** – Represents an order confirmation, which is sent to the IMAge ERP whenever a store order arrives to the ORMS.

With the implementation of this solution, stores that already operate with the IMAge solution will be able to create store orders to the warehouses and acknowledge the reception of the ordered goods. An overview schema of this whole EAI process is represented in Figure 4.15.

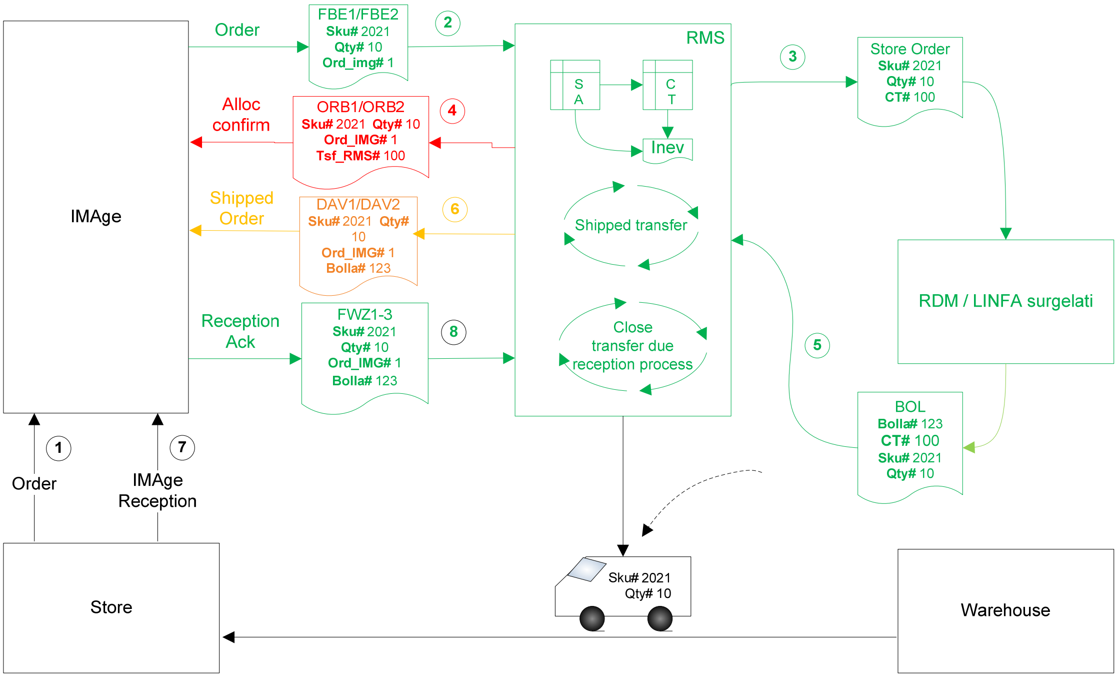


Figure 4.15: Shipments and Orders Responses from ORMS to IMAge

For now, the objective of this project involves the implementation of the IMAge subscribers for the ORMS published Shipment (DAV) and Order Response (ORB) data into the integration BUS, which are represented in Figure 4.15 by numbers 6 (yellow) and 4 (red), correspondingly.

4.4.1 Assumptions

The following assumptions will be taken into account for the development of the Store to Warehouse Orders and Receptions project:

- The EAI publication process will not include any data transformation.
- The data recovery will be manual.
- Each message family will use a different NB_EAI_<OBJECT> (Oracle Object Type). Any changes to the structure of the messages, will affect the specific message family and correspondent NB_EAI_<OBJECT>, the JCAPS OTD's and the Java Wrapper Classes.
- The external information block concept (EXT_INFO) will be only used as an exception measure.
- The ORMS system will ensure the messages publication in the correct order.
- There will be no message bundling; each message will be published individually.
- Delete and modification messages won't be implemented for the new families.
- There won't be more than one unpublished message for the same ID.

- There will be no message binding (process will not validate if a message was already published in the past).
- It is not in the scope of this project to develop any new alarm or monitoring mechanisms to process error messages. The current monitoring and error handling mechanisms will be used.

4.4.2 Architecture

In Figure 4.16 is detailed the architecture of the Store to Warehouse Orders and Receptions data integration flow to be implemented, giving a general overview of the components to develop, which are described in the subsequent sections.

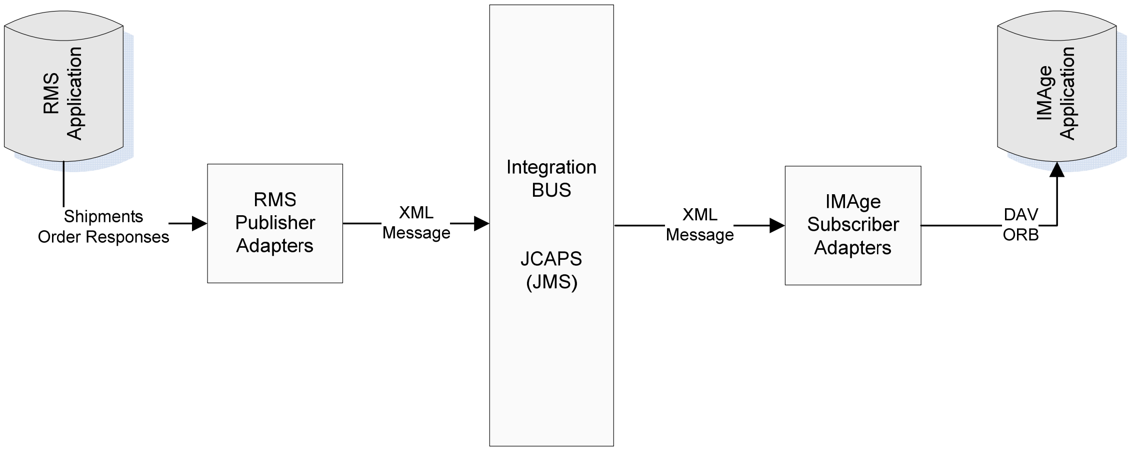


Figure 4.16: Store to Warehouse Orders and Receptions architecture

The integration BUS will be used to hold published data from applications. This way the ORMS application will send information about Warehouse Notifications, which include Shipments and Orders Responses data that will be integrated into IMAge using the mapping detailed in Table 4.15.

Table 4.15: Warehouse Notifications mapping table

Publication		*Subscription*
Family	ORMS	IMAge data
WhNotification	ShipmentFullDesc	DAV1
		DAV2
	OrderResponseFullDesc	ORB1
		ORB2

4.4.3 IMAge Subscribers

4.4.3.1 Shipment Subscriber

For the IMAge subscriber for Shipment (DAV) data the necessary Java Wrapper Classes have been created and added to a JCAPS project, in order to make possible for the framework to interchange data between applications, in this case between JCAPS and the IMAge system. Therefore, the Java Wrapper Classes are responsible for translating the data, previously formatted according to the OTD, *ShipmentFullDesc* (Appendix B.4), and received from the integration BUS, into the developed Oracle Object Types. The Java Wrapper Classes are then created using Oracle JDeveloper.

Regarding the IMAge DAV subscriber for ORMS Shipment data, it will pick the data from the integration BUS and map it according to the IMAge message specification, by using a mapping package designed for this effect. After invoking this mapping procedure, the message is then sent to an IMAge JMS input queue in order to be processed by this application.

The developed subscription project is implemented by a collaboration, *svcMapDAVToImage* (Appendix F.6), and uses the configuration described in Figure 4.12.

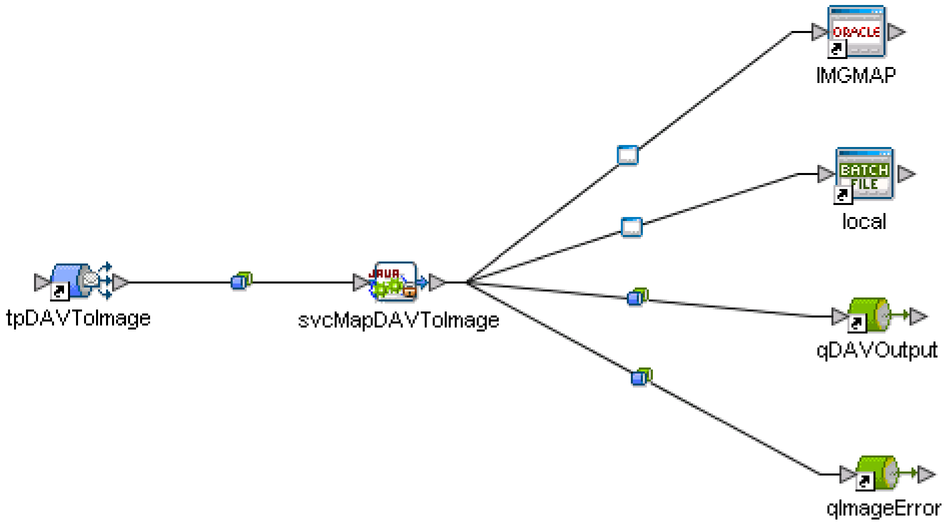


Figure 4.17: Shipment (DAV) IMAge subscriber connectivity map definition

The components represented in the figure above are described in Table 4.16.

Table 4.16: Shipment (DAV) IMAge subscriber components

Shipment (DAV) subscription components			
Object	Type	New?	Description
tpDAVToImage	JMS topic	Y	Contains the ORMS data formatted from the integration BUS.
svcMapDAVToImage	Java Collaboration	Y	It receives the CMF Shipment Message (*ShipmentFullDesc*) and calls the mapping to DAV IMAge message, which is then sent into the *qDAVOutput* JMS queue that would be subscribed afterwards by the IMAge.
IMGMAP	Oracle External Application	Y	Provides an Oracle database connection.
local	File adapter	Y	Allows access to the subscription properties file.
qDAVOutput	JMS queue	Y	DAV JMS output queue for all subscribed IMAge messages.
qImageError	JMS queue	Y	JMS queue that holds any eventual error message that occurred during the subscription process.

4.4.3.2 OrderResponse Subscriber

In relation to the OrderResponse (ORB) data subscription, the JCAPS project includes the Java Wrapper Classes, as in previous projects, to allow the framework to map data between systems, in this specific case between JCAPS and the IMAge system. Thus, the function of the Java Wrapper Classes is to translate the data, previously formatted according to the OTD, *OrderResponseFullDesc* (Appendix B.5), and received from the integration BUS, into the developed Oracle Object Types. After the creation of the Oracle Object Types, the Java Wrapper Classes are created using Oracle JDeveloper.

For the IMAge ORB subscriber for ORMS OrderResponse data, it will pick the data from the integration BUS and map it according to the IMAge message specification, by using a mapping package designed for this effect. After invoking this mapping procedure, the message is then sent to an IMAge JMS input queue in order to be processed by this application.

The subscriber is implemented by a collaboration, *svcORBToImage* (Appendix F.6), and uses the configuration described in Figure 4.11.

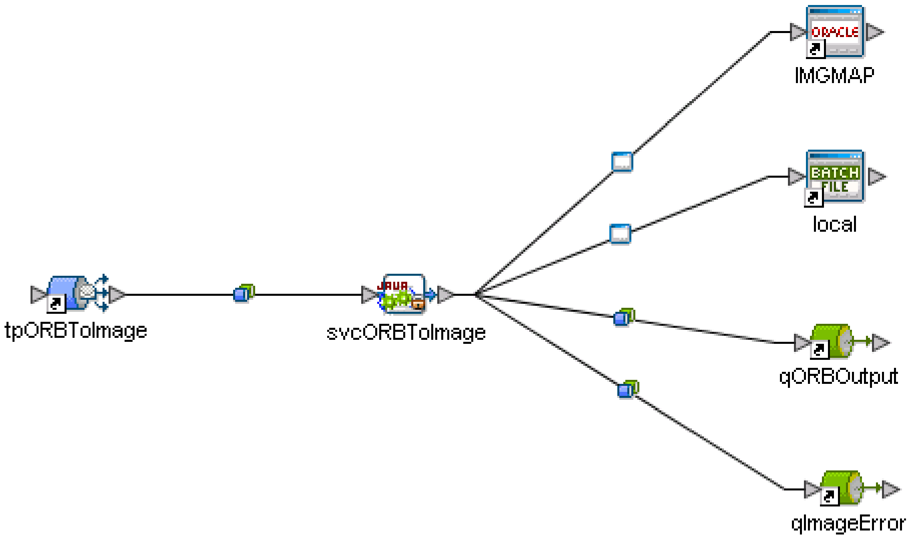


Figure 4.18: OrderResponse (ORB) IMAge subscriber connectivity map definition

The components used in the connectivity map and illustrated above, are detailed in Table 4.17.

Table 4.17: OrderResponse (ORB) IMAge subscriber components

OrderResponse (ORB) subscription components			
Object	Type	New?	Description
tpORBToImage	JMS topic	Y	Contains the ORMS data formatted from the integration BUS.
svcMapORBToImage	Java Collaboration	Y	It receives the CMF Shipment Message (*OrderResponseFullDesc*) and calls the mapping to ORB IMAge message, which is then sent into the *qORBOutput* JMS queue that would be subscribed afterwards by the IMAge.
IMGMAP	Oracle External Application	Y	Provides an Oracle database connection.
local	File adapter	Y	Allows access to the subscription properties file.
qORBOutput	JMS queue	Y	ORB JMS output queue for all subscribed IMAge messages.
qImageError	JMS queue	Y	JMS queue that holds any eventual error message that occurred during the subscription process.

4.4.4 Data Model

As previously described, this project comprises two business entities, Shipment and OrderResponse, which are generalized by the WhNotification entity. For the IMAge message specifications, the Shipments are mapped into the DAV (Dispatch Advice) message family and the Orders Responses into the ORB (Order Response) message family.

The DAV and ORB definition is divided in the following parts:

- **DAV1** – It is composed by the Shipments IMAge message header.
- **DAV2** – Formed by a logic of detail, which contains the IMAge message details associated with the Shipments header of the message.
- **ORB1** – Consists of the Orders Responses IMAge message header.
- **ORB2** – This is the logic of detail, associated with the Order Responses header, which includes the IMAge message details.

Thus, the DAV and ORB message families correspond to the Shipment and OrderResponse CMF. Each one of this CMF defines the ORMS message specification and are divided in the following parts:

- **ShipmentFullDesc** – It is composed by a header which contains the standard features of the entity, in this case shipped items description.
- **ShipSkuDesc** – This other part is formed by a logic of detail, to contain the Shipments' features data associated with the header of the message.
- **OrderResponseFullDesc** – This represents the standard features of the Order Response, which correspond to the header of the message.
- **OrderResponseDtlDesc** – Corresponds to the logic of detail relative to the Orders Responses, which is associated with the *OrderResponseFullDesc*.

The data mapping between the mentioned entities, Shipment to DAV, and OrderResponse to ORB, is made with the aid of a mapping package which is implemented according to the specific mapping needs of the involved messages. After the successful completion of the map procedure, the IMAge formatted data is sent to an input JMS queue for post-processing.

More information on the data model used in the Store to Warehouse Orders and Receptions project, namely the messages specification, is detailed in Appendixes B.4 to B.9.

4.4.5 IMAge Mapping Packages

These components are responsible for mapping the data, published into the integration BUS by the ORMS application, according to the IMAge input message format. The developed IMAge mapping packages will pick the data received as an Oracle Object Type and return a XML message in compliance with the IMAge message specification.

Since the presented mapping methodology is entirely implemented in an Oracle database, this approach uses rules stored in database tables to transform and create incoming data to the IMAge message format.

The data model where all the process lays is shown in Figure 4.19.

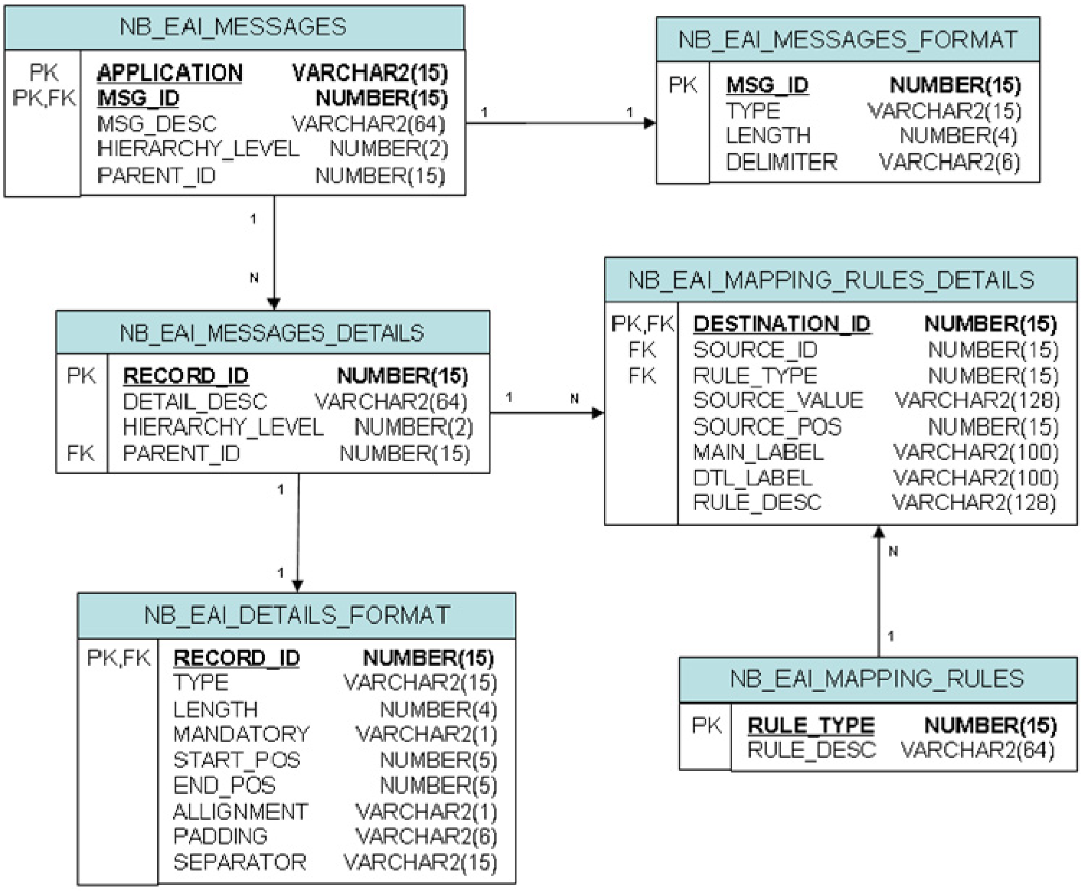


Figure 4.19: IMAge mapping rules data model

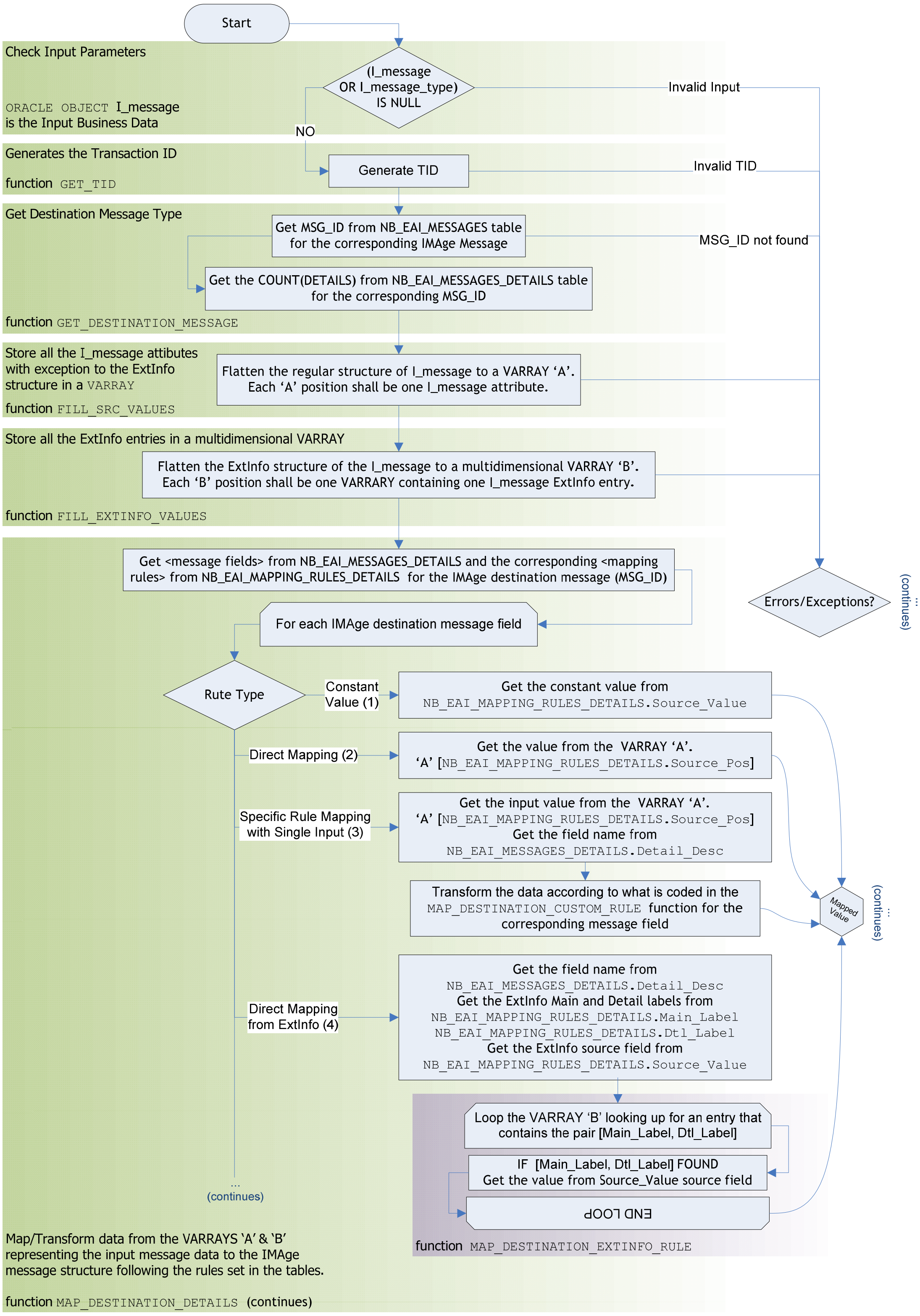
The IMAge mapping packages naming follows the usual practice adopted in this kind of EAI projects which is the NB_IMGMAP_<BUSINESS_ENTITY> naming convention. In this particular case, two packages have been developed – NB_IMGMAP_DAV and NB_IMGMAP_ORB – corresponding to the Shipment and OrderResponse business entities.

The packages procedures and functions are described in Table 4.18.

Table 4.18: IMAge DAV/ORB mapping packages description

IMAge DAV/ORB mapping package basic components	
Procedure/Function name	Description
BUILD_MESSAGE	Build the IMAge message by assembling the header and detail sections.
MAP	This procedure will be called by the JCAPS IMAge subscriber and returns the transformed message to IMAge application.
GET_TID	Generates the Transaction ID.
GET_MSG_TYPE	Determines the incoming message type.
GET_DESTINATION_MESSAGE	Determines the destination message ID.
FORMAT_FIELDS	Applies formatting rules to fields.
FILL_SRC_VALUES	Populates auxiliary structures for data mapping.
FILL_EXTINFO_VALUES	Populates ExtInfo auxiliary structures for data mapping.
MAP_DESTINATION_DETAILS	Applies mapping rules (mapping types) for each incoming message field.
MAP_DESTINATION_CUSTOM_RULE	Applies data transformation and custom mapping for message data fields.
MAP_DST_CUSTOM_MULTIPLE_RULES	Applies data transformation and custom mapping for message data fields.
MAP_DESTINATION_EXTINFO_RULE	Applies data transformation and custom mapping for message data fields that need information from ExtInfo data structure.

The logic implemented by the IMAge mapping packages is represented by the flowchart in Figure 4.20.


Start
Check Input Parameters
ORACLE OBJECT I_message is the Input Business Data
(I_message OR I_message_type) IS NULL
Invalid Input
NO
Generates the Transaction ID
function GET_TID
Generate TID
Invalid TID
Get Destination Message Type
Get MSG_ID from NB_EAI_MESSAGES table for the corresponding IMAge Message
MSG_ID not found
Get the COUNT(DETAILS) from NB_EAI_MESSAGES_DETAILS table for the corresponding MSG_ID
function GET_DESTINATION_MESSAGE
Store all the I_message attibutes with exception to the ExtInfo structure in a VARRAY
function FILL_SRC_VALUES
Flatten the regular structure of I_message to a VARRAY 'A'. Each 'A' position shall be one I_message attribute.
Store all the ExtInfo entries in a multidimensional VARRAY
Flatten the ExtInfo structure of the I_message to a multidimensional VARRAY 'B'. Each 'B' position shall be one VARRARY containing one I_message ExtInfo entry.
function FILL_EXTINFO_VALUES
Get <message fields> from NB_EAI_MESSAGES_DETAILS and the corresponding <mapping rules> from NB_EAI_MAPPING_RULES_DETAILS for the IMAge destination message (MSG_ID)
Errors/Exceptions?
... (continues)
For each IMAge destination message field
Rute Type
Constant Value (1)
Get the constant value from NB_EAI_MAPPING_RULES_DETAILS.Source_Value
Direct Mapping (2)
Get the value from the VARRAY 'A'. 'A' [NB_EAI_MAPPING_RULES_DETAILS.Source_Pos]
Specific Rule Mapping with Single Input (3)
Get the input value from the VARRAY 'A'. 'A' [NB_EAI_MAPPING_RULES_DETAILS.Source_Pos] Get the field name from NB_EAI_MESSAGES_DETAILS.Detail_Desc
Transform the data according to what is coded in the MAP_DESTINATION_CUSTOM_RULE function for the corresponding message field
Mapped Value
... (continues)
Direct Mapping from ExtInfo (4)
Get the field name from NB_EAI_MESSAGES_DETAILS.Detail_Desc Get the ExtInfo Main and Detail labels from NB_EAI_MAPPING_RULES_DETAILS.Main_Label NB_EAI_MAPPING_RULES_DETAILS.Dtl_Label Get the ExtInfo source field from NB_EAI_MAPPING_RULES_DETAILS.Source_Value
Loop the VARRAY 'B' looking up for an entry that contains the pair [Main_Label, Dtl_Label]
IF [Main_Label, Dtl_Label] FOUND Get the value from Source_Value source field
END LOOP
function MAP_DESTINATION_EXTINFO_RULE
... (continues)
Map/Transform data from the VARRAYS 'A' & 'B' representing the input message data to the IMAge message structure following the rules set in the tables.
function MAP_DESTINATION_DETAILS (continues)

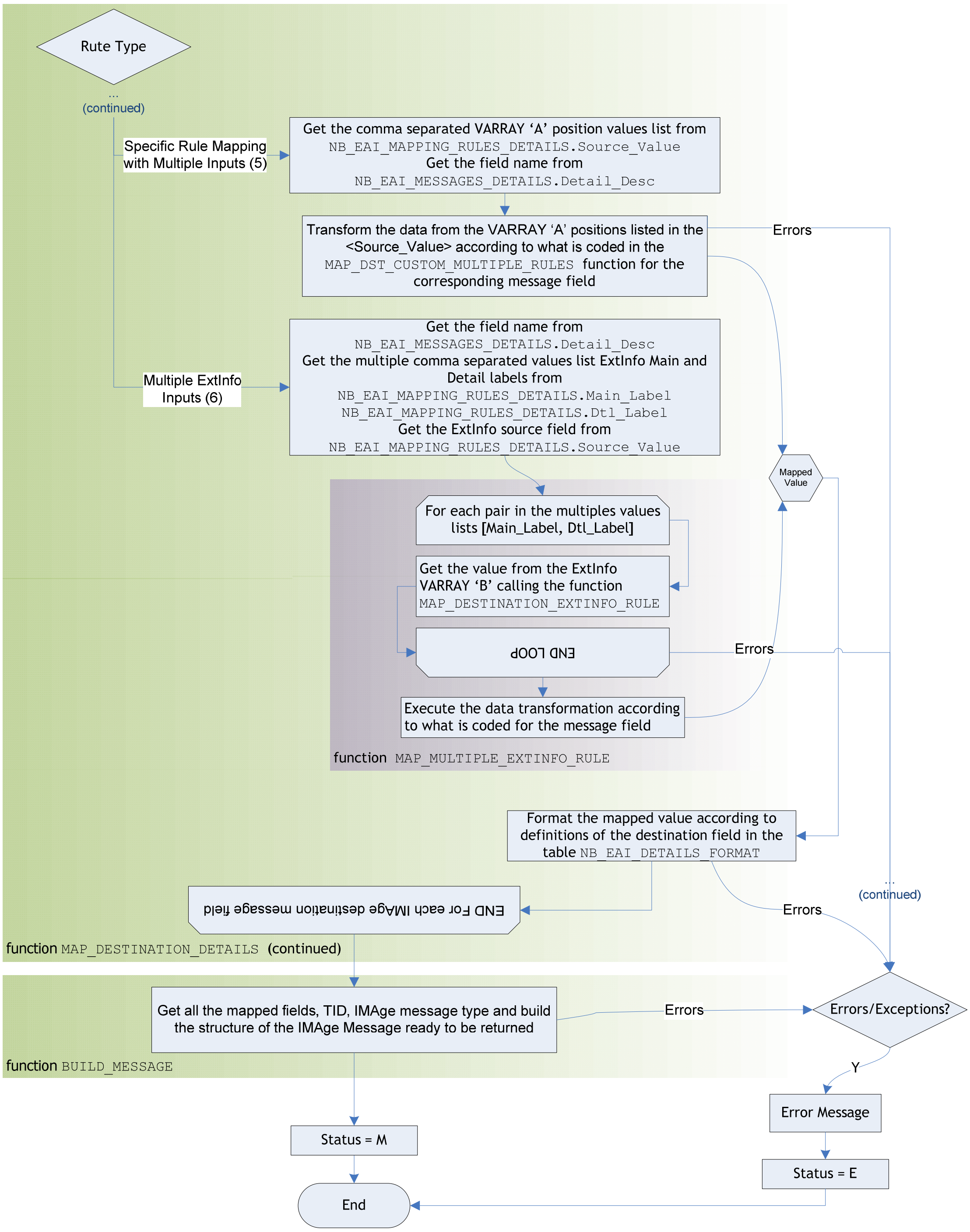

Rute Type
…
(continued)
Specific Rule Mapping with Multiple Inputs (5)
Get the comma separated VARRAY 'A' position values list from NB_EAI_MAPPING_RULES_DETAILS.Source_Value
Get the field name from NB_EAI_MESSAGES_DETAILS.Detail_Desc
Transform the data from the VARRAY 'A' positions listed in the <Source_Value> according to what is coded in the MAP_DST_CUSTOM_MULTIPLE_RULES function for the corresponding message field
Errors
Multiple ExtInfo Inputs (6)
Get the field name from NB_EAI_MESSAGES_DETAILS.Detail_Desc
Get the multiple comma separated values list ExtInfo Main and Detail labels from NB_EAI_MAPPING_RULES_DETAILS.Main_Label NB_EAI_MAPPING_RULES_DETAILS.Dtl_Label
Get the ExtInfo source field from NB_EAI_MAPPING_RULES_DETAILS.Source_Value
Mapped Value
For each pair in the multiples values lists [Main_Label, Dtl_Label]
Get the value from the ExtInfo VARRAY 'B' calling the function MAP_DESTINATION_EXTINFO_RULE
END LOOP
Errors
Execute the data transformation according to what is coded for the message field
function MAP_MULTIPLE_EXTINFO_RULE
Format the mapped value according to definitions of the destination field in the table NB_EAI_DETAILS_FORMAT
…
(continued)
END For each IMAge destination message field
Errors
function MAP_DESTINATION_DETAILS (continued)
Get all the mapped fields, TID, IMAge message type and build the structure of the IMAge Message ready to be returned
Errors
Errors/Exceptions?
function BUILD_MESSAGE
Y
Error Message
Status = M
Status = E
End

Figure 4.20: IMAge mapping packages algorithm

A more detailed description of the IMAge mapping packages, namely its procedures and functions, can be found in Appendix E.

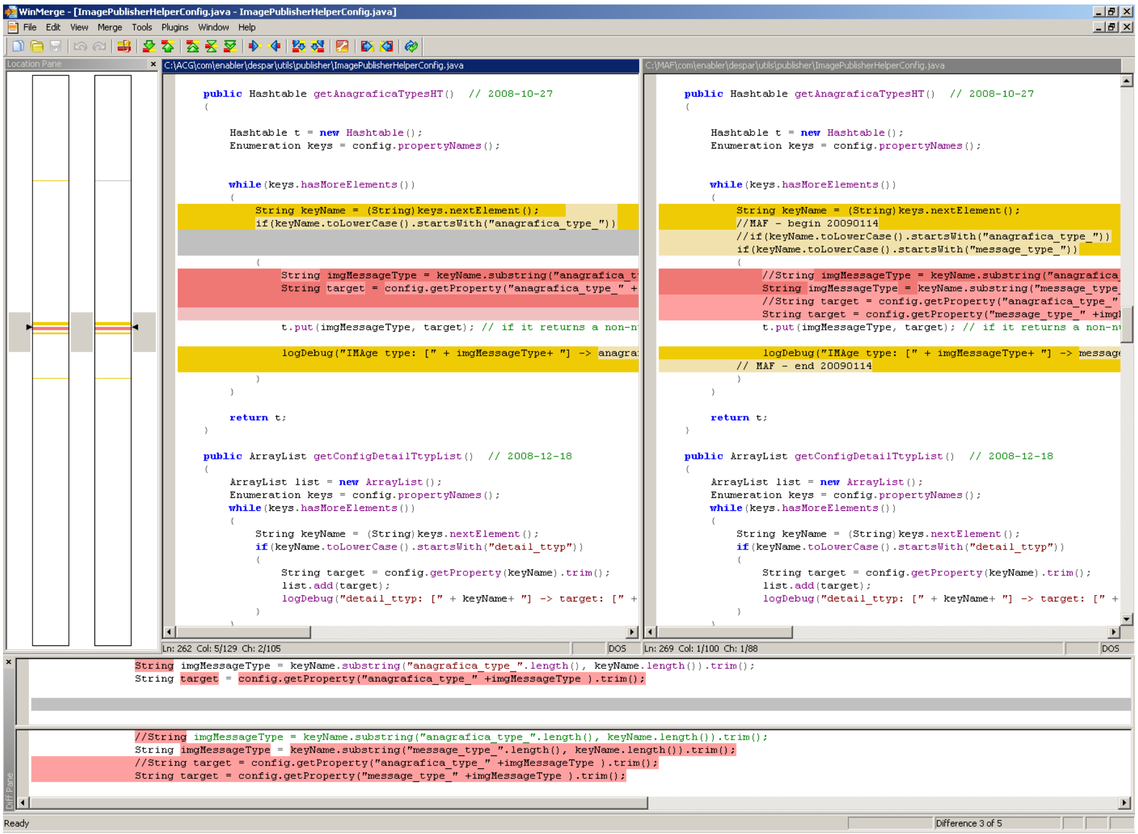
4.5 DeSpar Pub/Sub Framework Review

Being part of the DeSpar EAI project team, allowed for the author to have a general view of the retail business model and, at a lower level, an acquisition and consolidation of knowledge regarding the systems used by the client and their needs for integration.

However, during the developed work throughout the internship there was a component that was viewed by the author as a kind of "black box", although being aware of its functional context. This component is the DeSpar Pub/Sub framework.

The lack of technical knowledge of the framework was contradicted by a required review for different versions merging and, later on, an impact analysing of this version merge into the existing JCAPS Publisher/Subscriber projects. The objective is to analyse eventual differences between disparate versions of the framework which exist in development (DEV) environment, the machine used by Wipro Retail for implementation and testing. This is a very important work that needed to be done in order to know the latest stable version of the framework that should be used later in acceptance (ACC) environment, which corresponds to the machine used by the client for User Acceptance Testing (UAT).

The execution of the framework review task has been accomplished by using WinMerge, a free software tool for file comparison and merging text-like files. Figure 4.21 illustrates an example of differences found between disparate versions of the DeSpar Pub/Sub framework.


WinMerge - [ImagePublisherHelperConfig.java - ImagePublisherHelperConfig.java]
File Edit View Merge Tools Plugins Window Help
Location Pane
Diff Pane
Ready
Difference 3 of 5
Ln: 262 Col: 5/129 Ch: 2/105
DOS
Ln: 269 Col: 1/100 Ch: 1/88
DOS

C:\ACG\com\enabler\despar\utils\publisher\ImagePublisherHelperConfig.java

```
    public Hashtable getAnagraficaTypesHT()  // 2008-10-27
    {

        Hashtable t = new Hashtable();
        Enumeration keys = config.propertyNames();


        while(keys.hasMoreElements())
        {
            String keyName = (String)keys.nextElement();
            if(keyName.toLowerCase().startsWith("anagrafica_type_"))


            {
                String imgMessageType = keyName.substring("anagrafica_t
                String target = config.getProperty("anagrafica_type_" +


                t.put(imgMessageType, target); // if it returns a non-n

                logDebug("IMAge type: [" + imgMessageType+ "] -> anagra

            }
        }

        return t;
    }

    public ArrayList getConfigDetailTtypList()  // 2008-12-18
    {
        ArrayList list = new ArrayList();
        Enumeration keys = config.propertyNames();
        while(keys.hasMoreElements())
        {
            String keyName = (String)keys.nextElement();
            if(keyName.toLowerCase().startsWith("detail_ttyp"))
            {
                String target = config.getProperty(keyName).trim();
                list.add(target);
                logDebug("detail_ttyp: [" + keyName+ "] -> target: [" +
            }
```

C:\MAF\com\enabler\despar\utils\publisher\ImagePublisherHelperConfig.java

```
    public Hashtable getAnagraficaTypesHT()  // 2008-10-27
    {

        Hashtable t = new Hashtable();
        Enumeration keys = config.propertyNames();


        while(keys.hasMoreElements())
        {
            String keyName = (String)keys.nextElement();
            //MAF - begin 20090114
            //if(keyName.toLowerCase().startsWith("anagrafica_type_"))
            if(keyName.toLowerCase().startsWith("message_type_"))
            {
                //String imgMessageType = keyName.substring("anagrafica
                String imgMessageType = keyName.substring("message_type
                //String target = config.getProperty("anagrafica_type_"
                String target = config.getProperty("message_type_" +img
                t.put(imgMessageType, target); // if it returns a non-n

                logDebug("IMAge type: [" + imgMessageType+ "] -> message
            // MAF - end 20090114
            }
        }

        return t;
    }

    public ArrayList getConfigDetailTtypList()  // 2008-12-18
    {
        ArrayList list = new ArrayList();
        Enumeration keys = config.propertyNames();
        while(keys.hasMoreElements())
        {
            String keyName = (String)keys.nextElement();
            if(keyName.toLowerCase().startsWith("detail_ttyp"))
            {
                String target = config.getProperty(keyName).trim();
                list.add(target);
                logDebug("detail_ttyp: [" + keyName+ "] -> target: [" +
            }
```

```
String imgMessageType = keyName.substring("anagrafica_type_".length(), keyName.length()).trim();
String target = config.getProperty("anagrafica_type_" +imgMessageType ).trim();
```

```
//String imgMessageType = keyName.substring("anagrafica_type_".length(), keyName.length()).trim();
String imgMessageType = keyName.substring("message_type_".length(), keyName.length()).trim();
//String target = config.getProperty("anagrafica_type_" +imgMessageType ).trim();
String target = config.getProperty("message_type_" +imgMessageType ).trim();
```

Figure 4.21: DeSpar Pub/Sub framework version analysis example using WinMerge

The review of the disparate versions of the framework resulted in a list of changes that were enumerated in order to determine which of the versions currently being used in DEV environment better handles the Publication/Subscription requirements for the existing JCAPS projects. The differences between versions were found in the following components of the framework:

- **Error Handling** – Found changes in the OTD's used for error handling, namely in the error message definition.
- **Reflection Utilities** – These utilities take advantage of the Java Reflection API to read and write fields and methods of a selected class in runtime, namely the methods used to map the CMF messages into the Oracle Object Types. The detected changes were precisely in the way the framework accesses these methods.
- **Object Mapping** – Some features have been deprecated in these classes responsible for the mapping between CMF OTD's and Oracle Objects.
- **IMAge Publishers** – Changes that affected the properties specified in the configuration files used by the IMAge publishers.
- **Java Wrapper Classes** – The way the Java Wrapper Classes initialize the Oracle Objects was also subject of changes. A new Java interface class has been included to deal with the process of Oracle Objects initialization, allowing for a more flexible application decoupling.

With these versioning differences detected, the isolation of the latest stable version of the DeSpar Pub/Sub framework has been accomplished by importing different JAR versions of the framework into the projects in DEV environment and then analysing the impact this action had on the existing integration flows, by means of unit testing. Ultimately a stable version of the framework has been found, only requiring small changes in the IMAge publication configuration files and regeneration of the Java Wrapper Classes initializers.

This task is perfectly contextualized within the scope and objectives of this project since it allowed for a better understanding of the Publication/Subscription model and provided skills for developing future integration flows using an integration BUS framework.

5 Tests

In this chapter, the testing strategy, used to evaluate the developed integration solution, is detailed across the different test approaches for verification and validation of the various project developments.

5.1 Unit Testing

Unit tests check the individual software components during development. In computer programming, unit testing consists of a verification and validation method applied to a software unit, which represents the smallest testable part of an application.

For the implemented Publisher/Subscriber integration flows, using JCAPS, the unit tests have been performed throughout the development phase and in a Development (DEV) environment.

In general terms, the unit tests produced for a Publication/Subscription project should meet the following criteria:

- Message is received from the publisher
- Message is sent for publication into the integration BUS
- Message data is correctly mapped into the CMF
- Data is successfully subscribed
- Data is inserted into the staging tables

The unit tests produced for the developed work were made by the author and are presented in a Unit Test Document (UTD), which is then included in the documentation of the project. The UTD's produced for Store Productivity, Reception/RTV and Store to Warehouse Orders and Receptions projects are presented in Appendix G.

5.2 Integration Testing

The developed work throughout the internship is currently being subject to integration testing. Since the tests are still underway by a DeSpar development and testing team, no feedback can be reported for this kind of tests.

The goal of the integration tests is to validate the interface between software components. This sort of tests occurs after unit testing and before user acceptance testing, for the reason that the inputs for integration testing are modules which are supposed to have been unit tested.

Once these modules are grouped into a larger collection, tests that constitute an integration test plan are applied to those collections. The obtained output is an integrated system ready for User Acceptance Testing (UAT).

For the developed Publisher/Subscriber integration flows, the integration tests should verify and validate the integration with the DeSpar Pub/Sub framework and the involved applications.

5.2.1 DeSpar Pub/Sub Framework Integration Testing

It is intended with this testing phase to validate the integration between the two entities involved in a Publication/Subscription model of EAI – the Publisher and the Subscriber – and, in this particular case, the coordination between the Common Message Format (CMF) used in the publication and subscription phases of integration, by the DeSpar Pub/Sub framework.

These integration tests ensure that the inputs sent to the framework are in accordance with the CMF specification expected by the framework, as well as its outputs, which should be successfully mapped into the Oracle Objects with the aid of the Java Wrapper Classes. These framework inputs are tested during the publication phase, while its outputs correspond to the subscription phase.

The data mapping processes are also tested, namely in the comparison between the received messages format and the one specified in the CMF, and data validation.

5.2.2 Application Integration Testing

This is a more functional testing and is basically responsible for checking if the operations are successfully executed and data effectively exchanged between the applications involved in the Publisher/Subscriber integration flow.

For the integration flows implemented during the internship, the publishing applications being subject of testing are the AS400 system (for the Store Productivity application), the IMAge ERP (for the Reception/RTV change request) and the ORMS (for the Store to Warehouse Orders and Receptions). Both the first two projects have as subscribing application the ORMS, more specifically its staging tables, whose data is then internally processed and afterwards made available to be viewed in the ORMS interface. On the other hand, the Store to Warehouse Orders and Receptions project has for subscriber the IMAge application.

5.3 User Acceptance Testing

User Acceptance Testing (UAT) is the responsibility of DeSpar and is made subsequently by a client testing team, to check if the developed software meets its requirements. The developed work will be subject to this phase of testing, by the client, as soon as the integration tests are completed.

The UAT is one of the final stages of the project, and precedes the client acceptance of the new system or change request. These tests results are very important since they provide a certain degree of confidence to the client of the system performance in production.

6 Conclusions and Future Work

This final section concludes the report of the project, presenting a retrospective of the internship and evaluating the satisfaction of the proposed objectives. Some directions concerning future work are also expressed in this chapter.

6.1 Project Retrospective

The Enterprise Application Integration (EAI) is a complex, yet an indispensable process in the current world of business. The inexistence of a single software application capable of centralizing all business processes or the necessity for systems transition results in integration needs. For this reason, the motivation was always kept up very high during this project, supported by a personal interest by the author for the EAI area, especially due to its diversity in terms of distribution and integration technologies.

The project proved to be very challenging and enriching, allowing to have an overview of the retail market and its business entities. In this context, this project granted a familiarization with the main module of the Oracle Retail suite, the Oracle Retail Merchandising System (ORMS), from an integration point of view, as well as the proprietary system of the client, the IMAge. The study was focused on the message families supported by the publishers/subscribers and the integration of these messages into the DBMS supported by the mentioned systems. Also to emphasize, is the use of Java CAPS as a unified and comprehensive integration framework, which allows for an easy creation of composite applications, using a SOA approach.

For the developed work at Wipro Retail, this has always been in consonance with the project proposal and proved to be an extremely enriching experience, which was far beyond the acquisition of technical skills, also allowing for the development of team work capabilities. The clarification of doubts about any technology involved in the project has been as simple as possible, thanks to the people of the EAI team always available, thereby facilitating the sharing and acquisition of knowledge in the area.

Finally, at a personal and interpersonal level, the integration in Wipro Retail has been accomplished in the best way possible, much due to the excellent working conditions provided by the company, as well as a spirit of team work and mutual help, perceived throughout the internship.

6.2 Objectives Satisfaction

Given the date of beginning of the project and by the time the author start working at Wipro Retail, the phase of analysis and design of the integration solution to develop for this project was almost completed. Therefore, the work of the trainee was more targeted to a review of the proposed solution and its implementation and testing. This proved to be very advantageous because it allowed the acquisition of a much deeper technical expertise, given the technological variety present in an EAI project, while at the same time obtaining a sensibility for the integration needs of the client, thus acquiring the required skills to participate in the analysis of future integration interfaces for upcoming projects.

Regarding the objectives specified for this project, were overall achieved, except for the use of the Business Process Execution Language (BPEL), which was discarded in the phase of analysis and design, since its use was not justified given the business interactions involved in this project. However, the BPEL language is being subject of study by the client project team, in order to evaluate the use of this concept in future DeSpar projects and change requests.

In relation to the developed work, the best way to evaluate the objectives fulfilment is by the results obtained during the tests phase, particularly from the unit tests produced intensively to the implemented integration flows. These tests ensure that everything is perfectly functional as designed in the analysis of the integration solution. The testing covers all the interface of integration, from the publication to the subscription phase and subsequent database integration.

6.3 Future Work

With respect to future work concerning this project will be characterized by the migration of the developed integration interfaces into the acceptance (ACC) environment of the client. Once these integration flows are in production and validated by the client, other interfaces for different business entities will need to be developed as they are requested by DeSpar.

Also, as this internship made part of an Application Support and Maintenance (ASM) project signed-off between Wipro Retail and DeSpar, any change request that comes up from the client concerning Enterprise Application Integration (EAI) using Java CAPS should represent future work for the author.

One final advantage of this internship was the fact of conceding the necessary skills and competencies to develop integration solutions in the technological context of DeSpar, thus any present or future project, in the area of EAI, should represent potential work for the author.

References

[Aru08] Dhanasekar Arumugam. *Introduction to BPEL*. Wipro Technologies, November 2008.

[BPMN09] Object Management Group. Business Process Modeling Notation (BPMN) – Version 1.2, January 2009. http://www.omg.org/docs/formal/09-01-03.pdf .

[CKMS08] Michael Czapski, Sebastian Krueger, Brendan Marry, Saurabh Sahai, Peter Vaneris, and Andrew Walker. *Java CAPS Basics: Implementing Common EAI Patterns*. Prentice Hall, 2008.

[Des09] Despar Italia, 2009. http://www.desparitalia.it/ .

[Fer08] Marco Ferreira. *DeSpar Pub/Sub Framework ADSM Handover*. Wipro Retail, December 2008.

[Fer09a] Marco Ferreira. *DeSpar Pub/Sub Integration Flows – Mapping Packages Architecture for IMAge Subscribers*. Wipro Retail, March 2009.

[Fer09b] Marco Ferreira. *TRD 10774 – #1251 - IMAge - Store to WH Orders and Receptions*. Wipro Retail, June 2009.

[Goe08a] Henrique Goes. *Retail Today*. Wipro Retail, October 2008.

[Goe08b] Henrique Goes. *Oracle Retail (OR) Solutions Overview*. Wipro Retail, October 2008.

[Gui08] André Guimarães. *Sun Java CAPS 6 (Brief Overview)*. Wipro Retail, December 2008.

[Int08] InterSystems Corporation. SPAR Austria Empowers Local Store Managers with Caché-based ERP System, 2008. http://www.intersystems.com/casestudies/cache/spar.html .

[ISM08] *Introduction to SOA on Mainframes*. Wipro Technologies, 2008.

[Nar06] Lakshmi Narayanaswamy. *Possible Integration Patterns (White Paper)*. Wipro Technologies, December 2006.

[Ora09a] Oracle Corporation, 2009. http://www.oracle.com/ .

[Ora09b] Oracle Corporation. Oracle Retail Merchandising System Documentation Library, 2009. http://download.oracle.com/docs/cd/B31318_01/rms/index.html .

[Pei09a] Rui Peixoto. *TRD 10774 – #1203 - IMAge - Promotion New Requests implementation*. Wipro Retail, January 2009.

[Pei09b] Rui Peixoto. *TRD 10774 – #1227 - VEMO RTV interface implementation*. Wipro Retail, April 2009.

[PPCS09] Nuno Pinto, João Pinho, Paulo Correia, and Silvio Santos. *ASM DeSpar 2009 Presentation*. Wipro Retail, May 2009.

[Sha07] Hitesh Shah. *Message-Oriented Middleware*. Wipro Technologies, April 2007.

[SPAR09] SPAR International, 2009. http://www.spar-international.com/ .

[Sun08] Sun Microsystems, Inc. Sun Welcomes SeeBeyond Customers and Partners, 2008. http://www.sun.com/software/seebeyond/ .

[Sun09a] *Sun Java Composite Application Platform Suite (Java Caps) Data Sheet*. Sun Microsystems, Inc., 2009.

[Sun09b] Sun Microsystems, Inc. Sun Java Composite Application Platform Suite (Java CAPS), 2009. http://www.sun.com/software/javaenterprisesystem/javacaps/index.jsp .

[Sun09c] Sun Microsystems, Inc. Sun Java CAPS Documentation, 2009. http://developers.sun.com/docs/javacaps/api/javadocs/ .

[UTPG08] Unix Technology Practice Group, BFSI – TPG. *Enterprise Application Integration*. Wipro Technologies, October 2008.

[W3S09] W3Schools. DTD Tutorial, 2009. http://www.w3schools.com/dtd/default.asp .

[Wip09] Wipro Technologies. Retail, CPG & Distribution, 2009. http://www.wipro.com/retail/ .

[WRP08] *Wipro Retail Presentation*. Wipro Retail, 2008.

Appendix A: Publication Common Message Formats (CMF's)

This appendix contains the Common Message Format (CMF) specification for the messages used in the publication phase of the developed integration flows. Each CMF definition corresponds to an OTD (Object Type Definition) which is included in the correspondent publisher for the JCAPS project.

A.1 Store Productivity CMF

Table A.1: StoreProductivity

Field	*Destination CMF*	*CMF field*
DATE	StoreProdDesc	FILE_DATE
STORE	StoreProdDesc	STORE
REPARTO	StoreProdDtlDesc	REPARTO
HOUR_ORDINARY	StoreProdDtlDesc	NUM_HOURS
HOUR_EXTRA	StoreProdDtlDesc	NUM_HOURS_EXTRA
HOUR_VACATION	StoreProdDtlDesc	NUM_HOURS_VACATION
HOUR_ILL	StoreProdDtlDesc	NUM_HOURS_ILL

A.2 FWZ1 CMF

Table A.2: FWZ1

Field	*Destination CMF*	*CMF field*
FIL	ReceptionDesc/RTVDesc	LOCATION
WZNR	ReceptionDesc/RTVDesc	ACQUISITION_NR
BNR	ReceptionDesc/RTVDesc	TRANSACTION_NUMBER
BDATE	ReceptionDesc/RTVDesc	TRANSACTION_DATE
PWZDATE	ReceptionDesc/RTVDesc	PLAN_ACQUISITION_DATE
PWZZEIT	ReceptionDesc/RTVDesc	PLAN_ACQUISITION_SYSDATE
LINR	ReceptionDesc/RTVDesc	SUPPLIER
LSNR	ReceptionDesc/RTVDesc	RET_AUTH_NUM
LSDATE	ReceptionDesc/RTVDesc	ARRIVAL_DATE
WZDATE	ReceptionDesc/RTVDesc	SHIPMENT_DATE
PERSNR	ReceptionDesc/RTVDesc	USER_CODE
BART	ReceptionDesc/RTVDesc	PO_TYPE
WZINFO	ReceptionDesc	UPC_DESC
BOOKDATE	ReceptionDesc/RTVDesc	BOOKDATE
REFWZNR	ReceptionDesc	ACQUISITION_NR_REF
REFLSNR	ReceptionDesc	RET_AUTH_NUM_REF
REFLSDATE	ReceptionDesc	ARRIVAL_DATE_REF
EAN_CODE	ReceptionDesc/RTVDesc	UPC

A.3 FWZ2 CMF

Table A.3: FWZ2

Field	*Destination CMF*	*CMF field*
FIL	ReceptionDetailDesc	LOCATION
WZNR	ReceptionDetailDesc	ACQUISITION_NR
BNR	ReceptionDetailDesc	TRANSACTION_NUMBER
BPOS	ReceptionDetailDesc/RTVDetailDesc	TRANSACTION_SEQ_NO
ARTNR	ReceptionDetailDes	SKU

	c/RTVDetailDesc	
WZPOS	ReceptionDetailDesc/RTVDetailDesc	ACQUISITION_SKU
LAGER	ReceptionDetailDesc/RTVDetailDesc	WH
MGVE	ReceptionDetailDesc/RTVDetailDesc	SUPP_PACK_SIZE_QTY
FAKTOR	ReceptionDetailDesc/RTVDetailDesc	SUPP_PACK_SIZE
MEVE	ReceptionDetailDesc/RTVDetailDesc	QTY_RETURNED
NRABATT	ReceptionDetailDesc	DISCOUNT_TYPE
LISTPREIS	ReceptionDetailDesc	UNIT_COST_SUP_BASE
RABATT1	ReceptionDetailDesc	DISCOUNT_RATE_1
RABATT2	ReceptionDetailDesc	DISCOUNT_RATE_2
EKPREIS	ReceptionDetailDesc	UNIT_COST_ACTUAL
ESPREIS	ReceptionDetailDesc/RTVDetailDesc	UNIT_COST
KETTBASIS	ReceptionDetailDesc/RTVDetailDesc	CHAINING_BASE
KETTFACTOR	ReceptionDetailDesc/RTVDetailDesc	CHAINING_FACTOR
STATUS	ReceptionDetailDesc/RTVDetailDesc	STATUS
EKWERT	ReceptionDetailDesc/RTVDetailDesc	BUY_PRICE
MEVEGEBINDE	ReceptionDetailDesc/RTVDetailDesc	AUTO_SPK_QTY
MGVEGEBINDE	ReceptionDetailDesc/RTVDetailDesc	AUTO_RETURN_QTY
ARTTYP	ReceptionDetailDesc/RTVDetailDesc	ITEM_TYPE
BUCHUNGSLOGIK	ReceptionDetailDesc/RTVDetailDesc	RESERVATION_LOGIC
RICHTPREIS	ReceptionDetailDesc/RTVDetailDesc	UNIT_COST_DIRECT
LTISPREIVSOERRIDE	ReceptionDetailDesc	UNIT_COST_CONTROL
LIPOSNR	ReceptionDetailDesc	BOL_POSITION_NR

Appendix B: Subscription Common Message Formats (CMF's)

This appendix contains the Common Message Format (CMF) specification for the messages used in the subscription phase of the developed integration flows. Each CMF definition corresponds to an OTD (Object Type Definition) which is included in the correspondent subscriber for the JCAPS project.

B.1 Store Productivity CMF

Table B.1.1: StoreProdDesc

Field	*Destination table*	*Table field*	*Datatype*
FILENAME	NB_EAI_STORE_PROD_STG	FILENAME	VARCHAR2(30)
STORE	NB_EAI_STORE_PROD_STG	STORE	NUMBER(10)
FILE_DATE	NB_EAI_STORE_PROD_STG	FILE_DATE	VARCHAR2(20)
STORE_PROD_DTL_DESC*	NB_EAI_STORE_PROD_DTL_STG		

Table B.1.2: StoreProdDtlDesc

Field	*Destination table*	*Table field*	*Datatype*
REPARTO	NB_EAI_STORE_PROD_DTL_STG	REPARTO	NUMBER(15)
NUM_HOURS	NB_EAI_STORE_PROD_DTL_STG	NUM_HOURS	NUMBER(6,2)
NUM_HOURS_EXTRA	NB_EAI_STORE_PROD_DTL_STG	NUM_HOURS_EXTRA	NUMBER(6,2)
NUM_HOURS_VACATION	NB_EAI_STORE_PROD_DTL_STG	NUM_HOURS_VACATION	NUMBER(6,2)
NUM_HOURS_ILL	NB_EAI_STORE_PROD_DTL_STG	NUM_HOURS_ILL	NUMBER(6,2)

B.2 Reception CMF

Table B.2.1: ReceptionDesc

Field	*Destination table*	*Table field*	*Datatype*
TID	NB_EAI_RECEPTION_STG	TID	NUMBER(15)
LOCATION	NB_EAI_RECEPTION_STG	FELL	NUMBER(8)
ACQUISITION_NR	NB_EAI_RECEPTION_STG	WZNR	VARCHAR2(15)
TRANSACTION_NUMBER	NB_EAI_RECEPTION_STG	BNR	VARCHAR2(15)
TRANSACTION_DATE	NB_EAI_RECEPTION_STG	BDATE	DATE
PLAN_ACQUISITION_DATE	NB_EAI_RECEPTION_STG	PWZDATE	DATE
PLAN_ACQUISITION_SYSDATE	NB_EAI_RECEPTION_STG	PWZZEIT	DATE
SUPPLIER	NB_EAI_RECEPTION_STG	LINR	NUMBER(8)
RET_AUTH_NUM	NB_EAI_RECEPTION_STG	LSNR	VARCHAR2(15)
ARRIVAL_DATE	NB_EAI_RECEPTION_STG	LSDATE	DATE
SHIPMENT_DATE	NB_EAI_RECEPTION_STG	WZDATE	DATE
USER_CODE	NB_EAI_RECEPTION_STG	PERSNR	VARCHAR2(15)
PO_TYPE	NB_EAI_RECEPTION_STG	BEARD	NUMBER(2)
UPC_DESC	NB_EAI_RECEPTION_STG	WZINFO	VARCHAR2(200)
BOOKDATE	NB_EAI_RECEPTION_STG	BOOKDATE	DATE
ACQUISITION_NR_REF	NB_EAI_RECEPTION_STG	REFWZNR	VARCHAR2(15)
RET_AUTH_NUM_REF	NB_EAI_RECEPTION_STG	REFLSNR	VARCHAR2(15)
ARRIVAL_DATE_REF	NB_EAI_RECEPTION_STG	REFLSDATE	DATE
UPC	NB_EAI_RECEPTION_STG	EAN_CODE	VARCHAR2(16)
RECEPTIONDETAIL_DESC*	NB_EAI_RECEPTION_DETAIL_STG		

Table B.2.2: ReceptionDetailDesc

Field	*Destination table*	*Table field*	*Datatype*
LOCATION	NB_EAI_RECEPTION_DETAIL_STG	FELL	NUMBER(8)
ACQUISITON_NR	NB_EAI_RECEPTION_DETAIL_STG	WZNR	VARCHAR2(15)
TRANSACTION_NUMBER	NB_EAI_RECEPTION_DETAIL_STG	BNR	VARCHAR2(15)
TRANSACTION_SEQ_NO	NB_EAI_RECEPTION_DETAIL_STG	BPOS	NUMBER(4)
SKU	NB_EAI_RECEPTION_DETAIL_STG	ARTNR	NUMBER(8)
ACQUISITION_SKU	NB_EAI_RECEPTION_DETAIL_STG	WZPOS	NUMBER(4)
WH	NB_EAI_RECEPTION_DETAIL	CAMP	NUMBER(4)

	_STG		
SUPP_PACK_SIZE_QTY	NB_EAI_RECEPTION_DETAIL_STG	MGVE	NUMBER(8,4)
SUPP_PACK_SIZE	NB_EAI_RECEPTION_DETAIL_STG	FACTOR	NUMBER(10)
QTY_RETURNED	NB_EAI_RECEPTION_DETAIL_STG	MEVE	NUMBER(8,4)
DISCOUNT_TYPE	NB_EAI_RECEPTION_DETAIL_STG	NRABATT	VARCHAR2(1)
UNIT_COST_SUP_BASE	NB_EAI_RECEPTION_DETAIL_STG	CUNNING_PRICE	NUMBER(8)
DISCOUNT_RATE_1	NB_EAI_RECEPTION_DETAIL_STG	RABATT1	NUMBER(8)
DISCOUNT_RATE_2	NB_EAI_RECEPTION_DETAIL_STG	RABATT2	NUMBER(8)
UNIT_COST_ACTUAL	NB_EAI_RECEPTION_DETAIL_STG	EKPREIS	NUMBER(8)
UNIT_COST	NB_EAI_RECEPTION_DETAIL_STG	ESPREIS	NUMBER(8)
CHAINING_BASE	NB_EAI_RECEPTION_DETAIL_STG	CHAINING_BASIS	NUMBER(8)
CHAINING_FACTOR	NB_EAI_RECEPTION_DETAIL_STG	CHAINING_FACTOR	NUMBER(8)
STATUS	NB_EAI_RECEPTION_DETAIL_STG	STATUS	NUMBER(1)
BUY_PRICE	NB_EAI_RECEPTION_DETAIL_STG	EKWERT	NUMBER(20)
AUTO_SPK_QTY	NB_EAI_RECEPTION_DETAIL_STG	MEVEGEBINDE	NUMBER(8)
AUTO_RETURN_QTY	NB_EAI_RECEPTION_DETAIL_STG	MGVEGEBINDE	NUMBER(8)
ITEM_TYPE	NB_EAI_RECEPTION_DETAIL_STG	TYPE_OF_KIND	NUMBER(2)
RESERVATION_LOGIC	NB_EAI_RECEPTION_DETAIL_STG	RESERVATION_LOGIC	NUMBER(2)
UNIT_COST_DIRECT	NB_EAI_RECEPTION_DETAIL_STG	RICHTPREIS	NUMBER(12)
UNIT_COST_CONTROL	NB_EAI_RECEPTION_DETAIL_STG	LTISPREIVSOERRID	NUMBER(1)
BOL_POSITION_NR	NB_EAI_RECEPTION_DETAIL_STG	LIPOSNR	VARCHAR2(6)

B.3 RTV CMF

Table B.3.1: RTVDesc

Field	*Destination table*	*Table field*	*Datatype*
TID	NB_EAI_RTV_STG	TID	NUMBER(13)
TRANSACTION_NUMBER	NB_EAI_RTV_STG	RTV_ORDER_NO	NUMBER(6)
SUPPLIER	NB_EAI_RTV_STG	SUPPLIER	NUMBER(10)
STATUS_IND	NB_EAI_RTV_STG	STATUS_IND	NUMBER(2)
LOCATION	NB_EAI_RTV_STG	STORE	NUMBER(4)
WH	NB_EAI_RTV_STG	WH	NUMBER(4)

TOTAL_ORDER_AMT	NB_EAI_RTV_STG	TOTAL_ORDER_AMT	NUMBER(20,4)
SHIP_TO_ADD_1	NB_EAI_RTV_STG	SHIP_TO_ADD_1	VARCHAR2(30)
SHIP_TO_ADD_2	NB_EAI_RTV_STG	SHIP_TO_ADD_2	VARCHAR2(30)
SHIP_TO_ADD_3	NB_EAI_RTV_STG	SHIP_TO_ADD_3	VARCHAR2(30)
SHIP_TO_CITY	NB_EAI_RTV_STG	SHIP_TO_CITY	VARCHAR2(20)
STATE	NB_EAI_RTV_STG	STATE	VARCHAR2(3)
SHIP_TO_COUNTRY_ID	NB_EAI_RTV_STG	SHIP_TO_COUNTRY_ID	VARCHAR2(3)
SHIP_TO_PCODE	NB_EAI_RTV_STG	SHIP_TO_PCODE	VARCHAR2(10)
RET_AUTH_NUM	NB_EAI_RTV_STG	RET_AUTH_NUM	VARCHAR2(12)
COURIER	NB_EAI_RTV_STG	COURIER	VARCHAR2(20)
FREIGHT	NB_EAI_RTV_STG	FREIGHT	NUMBER(20,4)
CREATED_DATE	NB_EAI_RTV_STG	CREATED_DATE	DATE
COMPLETED_DATE	NB_EAI_RTV_STG	COMPLETED_DATE	DATE
HANDLING_PCT	NB_EAI_RTV_STG	HANDLING_PCT	NUMBER(12,4)
HANDLING_COST	NB_EAI_RTV_STG	HANDLING_COST	NUMBER(20,4)
EXT_REF_NO	NB_EAI_RTV_STG	EXT_REF_NO	VARCHAR2(14)
COMMENT_DESC	NB_EAI_RTV_STG	COMMENT_DESC	VARCHAR2(255)
ACQUISITION_NR	NB_EAI_RTV_STG	ACQUISITION_NR	VARCHAR2(15)
TRANSACTION_DATE	NB_EAI_RTV_STG	TRANSACTION_DATE	DATE
PLAN_ACQUISITION_DATE	NB_EAI_RTV_STG	PLAN_ACQUISITION_DATE	DATE
PLAN_ACQUISITION_SYSDATE	NB_EAI_RTV_STG	PLAN_ACQUISITION_SYSDATE	DATE
ARRIVAL_DATE	NB_EAI_RTV_STG	ARRIVAL_DATE	DATE
SHIPMENT_DATE	NB_EAI_RTV_STG	SHIPMENT_DATE	DATE
USER_CODE	NB_EAI_RTV_STG	USER_CODE	VARCHAR2(15)
PO_TYPE	NB_EAI_RTV_STG	PO_TYPE	NUMBER(2)
BOOKDATE	NB_EAI_RTV_STG	BOOKDATE	DATE
UPC	NB_EAI_RTV_STG	UPC	VARCHAR2(16)
RTVDETAIL_DESC*	NB_EAI_RTV_DETAIL_STG		

Table B.3.2: RTVDetailDesc

Field	*Destination table*	*Table field*	*Datatype*
SKU	NB_EAI_RTV_DETAIL_STG	SKU	NUMBER(8)
SHIPMENT	NB_EAI_RTV_DETAIL_STG	SHIPMENT	NUMBER(10)
INV_STATUS	NB_EAI_RTV_DETAIL_STG	INV_STATUS	NUMBER(2)
QTY_RETURNED	NB_EAI_RTV_DETAIL_STG	QTY_RETURNED	NUMBER(12,4)
UNIT_COST	NB_EAI_RTV_DETAIL_STG	UNIT_COST	NUMBER(20,4)
REASON	NB_EAI_RTV_DETAIL_STG	REASON	VARCHAR2(1)
TRANSACTION_SEQ_NO	NB_EAI_RTV_DETAIL_STG	RTV_SEQ_NO	NUMBER(4)
ACQUISITION_SKU	NB_EAI_RTV_DETAIL_STG	ACQUISITION_SKU	NUMBER(4)
SUPP_PACK_SIZE_QTY	NB_EAI_RTV_DETAIL_STG	SUPP_PACK_SIZE_QTY	NUMBER(8)
SUPP_PACK_SIZE	NB_EAI_RTV_DETAIL_STG	SUPP_PACK_SIZE	NUMBER(10)
CHAINING_BASE	NB_EAI_RTV_DETAIL_STG	CHAINING_BASE	NUMBER(8)
CHAINING_FACTOR	NB_EAI_RTV_DETAIL_STG	CHAINING_FACTOR	NUMBER(10)
STATUS	NB_EAI_RTV_DETAIL_STG	STATUS	NUMBER(1)
BUY_PRICE	NB_EAI_RTV_DETAIL_STG	BUY_PRICE	NUMBER(20)
AUTO_SPK_QTY	NB_EAI_RTV_DETAIL_STG	AUTOMATIC_SPK_QTY	NUMBER(8)
AUTO_RETURN_QTY	NB_EAI_RTV_DETAIL_STG	AUTOMATIC_RETURN_QTY	NUMBER(8)
ITEM_TYPE	NB_EAI_RTV_DETAIL_STG	ITEM_TYPE	NUMBER(2)
RESERVATION_LOGIC	NB_EAI_RTV_DETAIL_STG	RESERVATION_LOGIC	NUMBER(1)
UNIT_COST_CONTROL	NB_EAI_RTV_DETAIL_STG	UNIT_COST_CONTROL	NUMBER(1)

B.4 Shipment CMF

Table B.4.1: ShipmentFullDesc

Field	*Datatype*
SHIPMENT	NUMBER(10)
ORDER_NO	NUMBER(8)
TSF_NO	NUMBER(8)
SHIP_DATE	DATE
RECEIVE_DATE	DATE
EST_ARR_DATE	DATE
SHIP_ORIGIN	VARCHAR2(1)
STATUS_CODE	VARCHAR2(1)
INVC_MATCH_STATUS	VARCHAR2(1)
INVC_MATCH_DATE	DATE
TO_LOCATION	NUMBER(4)
TO_LOC_TYPE	VARCHAR2(1)
COURIER	VARCHAR2(20)
NO_BOXES	NUMBER(4)
QC_IND	VARCHAR2(1)

EXT_SHIPMENT	VARCHAR2(15)
EXT_REF_NO_IN	VARCHAR2(15)
EXT_REF_NO_OUT	VARCHAR2(15)
COMMENTS	VARCHAR2(160)
BARCODE	VARCHAR2(20)
FAIL_OCR_F	VARCHAR2(1)
OK_CM_F	VARCHAR2(1)
FROM_LOC_TYPE	VARCHAR2(1)
FROM_LOCATION	NUMBER(4)
SUPPLIER	NUMBER(10)
EXT_ORDER_NO	VARCHAR2(14)
SHIP_SKU_DESC*	

Table B.4.2: ShipSkuDesc

Field	*Datatype*
SHIPMENT	NUMBER(10)
SKU	NUMBER(8)
UPC	VARCHAR2(13)
UPC_SUPPLEMENT	NUMBER(5)
CARTON	VARCHAR2(20)
INV_STATUS	NUMBER(2)
STATUS_CODE	VARCHAR2(1)
QTY_RECEIVED	NUMBER(12,4)
UNIT_COST	NUMBER(20,4)
UNIT_RETAIL	NUMBER(20,4)
QTY_EXPECTED	NUMBER(12,4)
MATCH_INVC_ID	NUMBER(10)
QTY_SHIPPED	NUMBER(12,4)
SUPP_PACK_SIZE	NUMBER(4)
INNER_PACK_SIZE	NUMBER(4)
SHIP_CARTON_WT	NUMBER(12,4)
CATCH_WGH_F	VARCHAR2(1)
ITEM_TYPE_F	VARCHAR2(1)

B.5 OrderResponse CMF

Table B.5.1: OrderResponseFullDesc

Field	*Datatype*
EXT_ORDER_NO	VARCHAR2(14)
FROM_LOC	NUMBER(4)
FROM_LOC_TYPE	VARCHAR2(1)
TO_LOC	NUMBER(4)
TO_LOC_TYPE	VARCHAR2(1)
CURRENT_SYSDATE	DATE
SUPPLIER	NUMBER(10)
SUP_NAME	VARCHAR2(32)
CONTACT_PHONE	VARCHAR2(20)
CONTACT_FAX	VARCHAR2(20)
CONTACT_EMAIL	VARCHAR2(100)
ORDERRESPONSE_DTL_DESC*	

Table B.5.2: OrderResponseDtlDesc

Field	*Datatype*
EXT_ORDER_NO	VARCHAR2(14)
FROM_LOC	NUMBER(4)
FROM_LOC_TYPE	VARCHAR2(1)
TO_LOC	NUMBER(4)
TO_LOC_TYPE	VARCHAR2(1)
SKU	NUMBER(8)
SHIP_CARTON_WT	NUMBER(12,4)
INNER_PACK_SIZE	NUMBER(4)
TSF_QTY	NUMBER(12,4)
TSF_NO	NUMBER(8)
TSF_TYPE	VARCHAR2(2)

B.6 DAV1 CMF

Table B.6: DAV1

Field	*Source CMF*	*CMF field*
FIL	ShipmentFullDesc	TO_LOCATION
BNR	ShipmentFullDesc	TSF_NO
LINR	ShipmentFullDesc	SUPPLIER
LSNR	ShipmentFullDesc	EXT_SHIPMENT
LSDATE	ShipmentFullDesc	SHIP_DATE
LIEFDATE	ShipmentFullDesc	EST_ARR_DATE
TEILLIEF		(NULL)
ABSCHLUSS		(NULL)
AUFTYP		(0)

B.7 DAV2 CMF

Table B.7: DAV2

Field	*Source CMF*	*CMF field*
FIL	ShipmentFullDesc	TO_LOCATION
BNR	ShipmentFullDesc	TSF_NO
POS		(NULL)
ARTNR	ShipSkuDesc	SKU
MGVE	ShipSkuDesc	QTY_SHIPPED/SUPP_PACK_SIZE
MEVE	ShipSkuDesc	QTY_SHIPPED
EKPREIS	ShipSkuDesc	UNIT_COST
POSTYP	ShipSkuDesc	(CUSTOM_RULE)
POSABSCHLUSS		(NULL)
RICHTPREIS	ShipSkuDesc	UNIT_RETAIL
LIPOSNR		(NULL)
FAKTOR	ShipSkuDesc	(CUSTOM_MULTIPLE_RULE)
EAN	ShipSkuDesc	UPC

B.8 ORB1 CMF

Table B.8: ORB1

Field	*Source CMF*	*CMF field*
FIL	OrderResponseFullDesc	TO_LOC
BNR	OrderResponseDtlDesc	TSF_NO
ORDATE	OrderResponseFullDesc	CURRENT_SYSDATE
ORTIME	OrderResponseFullDesc	CURRENT_SYSDATE
LIEFNAME	OrderResponseFullDesc	SUP_NAME
LIEFTELEFON	OrderResponseFullDesc	CONTACT_PHONE
LIEFFAX	OrderResponseFullDesc	CONTACT_FAX
LIEFEMAIL	OrderResponseFullDesc	CONTACT_EMAIL
ORTYP	OrderResponseFullDesc	TSF_TYPE
LIEFNR	OrderResponseFullDesc	SUPPLIER

B.9 ORB2 CMF

Table B.9: ORB2

Field	*Source CMF*	*CMF field*
FIL	OrderResponseDtlDesc	TO_LOC
BNR	OrderResponseDtlDesc	TSF_NO
BESTPOS		(NULL)
ARTNR	OrderResponseDtlDesc	SKU
MGVE	OrderResponseDtlDesc	TSF_QTY/INNER_PACK_SIZE
MEVE	OrderResponseDtlDesc	TSF_QTY
BESTELLGRUND		(NULL)
BESTELLGRTEXT		(NULL)

Appendix C: Staging Tables

The staging tables, created to store temporary data, are detailed in this appendix. These tables are responsible for holding the mentioned temporary data before committing it to the final ORMS tables.

C.1 Store Productivity Staging Tables

Table C.1.1: NB_EAI_STORE_PROD_STG table

Primary key:	SEQ_NO, FILENAME, STORE, FILE_DATE		
Foreign keys:	N/A		
Fields			
Name	*Datatype*	*Null?*	*Default*
SEQ_NO	NUMBER(15)	No	
FILENAME	VARCHAR2(30)	No	
STORE	NUMBER(10)	No	
FILE_DATE	VARCHAR2(20)	No	
CREATION_SEQ_NO	NUMBER(15)	Yes	0
MESSAGE_TYPE	VARCHAR2(15)	No	
MESSAGE_SUBTYPE	VARCHAR2(15)	Yes	
TRANSACTION_TIME_STAMPS	DATE	Yes	
DATE_TO_BE_PROCESSED	DATE	No	
SUB_STATUS	VARCHAR2(1)	No	
ERROR_DESC	VARCHAR2(255)	Yes	

Table C.1.2: NB_EAI_STORE_PROD_DTL_STG table

Primary key:	SEQ_NO		
Foreign keys:	NB_EAI_STORE_PROD_STG.SEQ_NO		
Fields			
Name	*Datatype*	*Null?*	*Default*
SEQ_NO	NUMBER(15)	No	
REPARTO	NUMBER(15)	No	
NUM_HOURS	NUMBER(6,2)	Yes	
NUM_HOURS_EXTRA	NUMBER(6,2)	Yes	
NUM_HOURS_VACATION	NUMBER(6,2)	Yes	
NUM_HOURS_ILL	NUMBER(6,2)	Yes	

C.2 Reception Staging Tables

Table C.2.1: NB_EAI_RECEPTION_STG table

Primary key:	SEQ_NO		
Foreign keys:	N/A		
Fields			
Name	*Datatype*	*Null?*	*Default*
SEQ_NO	NUMBER(15)	No	
TID	NUMBER(15)	No	
FELL	NUMBER(8)	No	
WZNR	VARCHAR2(15)	No	
BNR	VARCHAR2(15)	No	
BDATE	DATE	No	
PWZDATE	DATE	No	
PWZZEIT	DATE	No	
LINR	NUMBER(8)	No	
LSNR	VARCHAR2(15)	No	
LSDATE	DATE	No	
WZDATE	DATE	No	
PERSNR	VARCHAR2(15)	No	
BEARD	NUMBER(2)	Yes	
WZINFO	VARCHAR2(200)	Yes	
BOOKDATE	DATE	No	
REFWZNR	VARCHAR2(15)	Yes	
REFLSNR	VARCHAR2(15)	Yes	
REFLSDATE	DATE	Yes	
EAN_CODE	VARCHAR2(16)	No	
USER_ID	VARCHAR2(30)	No	
MESSAGE_TYPE	VARCHAR2(15)	No	
MESSAGE_SUBTYPE	VARCHAR2(15)	Yes	
TRANSACTION_TIME_STAMPS	DATE	Yes	
DATE_TO_BE_PROCESSED	DATE	No	
SUB_STATUS	VARCHAR2(1)	No	
ERROR_DESC	VARCHAR2(255)	Yes	
RETRY_NO	NUMBER(8)	No	0

Table C.2.2: NB_EAI_RECEPTION_DETAIL_STG table

Primary key:	SEQ_NO		
Foreign keys:	N/A		
Fields			
Name	*Datatype*	*Null?*	*Default*
SEQ_NO	NUMBER(15)	No	
FELL	NUMBER(8)	No	
WZNR	VARCHAR2(15)	No	
BNR	VARCHAR2(15)	No	
BPOS	NUMBER(4)	No	
ARTNR	NUMBER(8)	No	
WZPOS	NUMBER(4)	No	
CAMP	NUMBER(4)	No	
MGVE	NUMBER(8,4)	Yes	
FACTOR	NUMBER(10)	No	
MEVE	NUMBER(8,4)	No	
NRABATT	VARCHAR2(1)	Yes	
CUNNING_PRICE	NUMBER(8)	Yes	
RABATT1	NUMBER(8)	Yes	
RABATT2	NUMBER(8)	Yes	
EKPREIS	NUMBER(8)	Yes	
ESPREIS	NUMBER(8)	Yes	
CHAINING_BASIS	NUMBER(8)	Yes	
CHAINING_FACTOR	NUMBER(8)	Yes	
STATUS	NUMBER(1)	No	
EKWERT	NUMBER(20)	No	
MEVEGEBINDE	NUMBER(8)	No	
MGVEGEBINDE	NUMBER(8)	No	
TYPE_OF_KIND	NUMBER(2)	Yes	
RESERVATION_LOGIC	NUMBER(2)	Yes	
RICHTPREIS	NUMBER(12)	Yes	
LTISPREIVSOERRID	NUMBER(1)	No	
LIPOSNR	VARCHAR2(6)	Yes	
USER_ID	VARCHAR2(30)	No	
MESSAGE_TYPE	VARCHAR2(15)	No	
MESSAGE_SUBTYPE	VARCHAR2(15)	Yes	
TRANSACTION_TIME_STAMPS	DATE	Yes	
DATE_TO_BE_PROCESSED	DATE	No	
SUB_STATUS	VARCHAR2(1)	No	
ERROR_DESC	VARCHAR2(255)	Yes	

C.3 RTV Staging Tables

Table C.3.1: NB_EAI_RTV_STG table

Primary key:	SEQ_NO
Foreign keys:	N/A

Fields

Name	*Datatype*	*Null?*	*Default*
SEQ_NO	NUMBER(15)	No	
TID	NUMBER(13)	No	
RTV_ORDER_NO	NUMBER(6)	No	
SUPPLIER	NUMBER(10)	No	
STATUS_IND	NUMBER(2)	Yes	
STORE	NUMBER(4)	Yes	
WH	NUMBER(4)	Yes	
TOTAL_ORDER_AMT	NUMBER(20,4)	Yes	
SHIP_TO_ADD_1	VARCHAR2(30)	Yes	
SHIP_TO_ADD_2	VARCHAR2(30)	Yes	
SHIP_TO_ADD_3	VARCHAR2(30)	Yes	
SHIP_TO_CITY	VARCHAR2(20)	Yes	
STATE	VARCHAR2(3)	Yes	
SHIP_TO_COUNTRY_ID	VARCHAR2(3)	Yes	
SHIP_TO_PCODE	VARCHAR2(10)	Yes	
RET_AUTH_NUM	VARCHAR2(12)	Yes	
COURIER	VARCHAR2(20)	Yes	
FREIGHT	NUMBER(20,4)	Yes	
CREATED_DATE	DATE	Yes	
COMPLETED_DATE	DATE	Yes	
HANDLING_PCT	NUMBER(12,4)	Yes	
HANDLING_COST	NUMBER(20,4)	Yes	
EXT_REF_NO	VARCHAR2(14)	Yes	
COMMENT_DESC	VARCHAR2(255)	Yes	
ACQUISITION_NR	VARCHAR2(15)	Yes	
TRANSACTION_DATE	DATE	Yes	
PLAN_ACQUISITION_DATE	DATE	Yes	
PLAN_ACQUISITION_SYSDATE	DATE	Yes	
ARRIVAL_DATE	DATE	Yes	
SHIPMENT_DATE	DATE	Yes	
USER_CODE	VARCHAR2(15)	Yes	
PO_TYPE	NUMBER(2)	Yes	
BOOKDATE	DATE	Yes	
UPC	VARCHAR2(16)	Yes	
USER_ID	VARCHAR2(30)	No	
MESSAGE_TYPE	VARCHAR2(15)	No	
MESSAGE_SUBTYPE	VARCHAR2(15)	Yes	
TRANSACTION_TIME_STAMPS	DATE	No	
DATE_TO_BE_PROCESSED	DATE	No	
SUB_STATUS	VARCHAR2(1)	No	
ERROR_DESC	VARCHAR2(255)	Yes	
RETRY_NO	NUMBER(8)	Yes	0

Table C.3.2: NB_EAI_RTV_DETAIL_STG table

Primary key:	SEQ_NO		
Foreign keys:	N/A		
Fields			
Name	*Datatype*	*Null?*	*Default*
SEQ_NO	NUMBER(15)	No	
RTV_ORDER_NO	NUMBER(6)	No	
SKU	NUMBER(8)	Yes	
SHIPMENT	NUMBER(10)	Yes	
INV_STATUS	NUMBER(2)	Yes	
QTY_RETURNED	NUMBER(12,4)	Yes	
UNIT_COST	NUMBER(20,4)	Yes	
REASON	VARCHAR2(1)	Yes	
RTV_SEQ_NO	NUMBER(4)	Yes	
ACQUISITION_SKU	NUMBER(4)	Yes	
SUPP_PACK_SIZE_QTY	NUMBER(8)	Yes	
SUPP_PACK_SIZE	NUMBER(10)	Yes	
CHAINING_BASE	NUMBER(8)	Yes	
CHAINING_FACTOR	NUMBER(10)	Yes	
STATUS	NUMBER(1)	Yes	
BUY_PRICE	NUMBER(20)	Yes	
AUTOMATIC_SPK_QTY	NUMBER(8)	Yes	
AUTOMATIC_RETURN_QTY	NUMBER(8)	Yes	
ITEM_TYPE	NUMBER(2)	Yes	
RESERVATION_LOGIC	NUMBER(2)	Yes	
UNIT_COST_CONTROL	NUMBER(1)	Yes	
USER_ID	VARCHAR2(30)	No	
MESSAGE_TYPE	VARCHAR2(15)	No	
MESSAGE_SUBTYPE	VARCHAR2(15)	Yes	
TRANSACTION_TIME_STAMPS	DATE	No	
DATE_TO_BE_PROCESSED	DATE	No	
SUB_STATUS	VARCHAR2(1)	No	
ERROR_DESC	VARCHAR2(255)	Yes	

Appendix D: ORMS Subscription Packages

This appendix presents a more detailed description of the ORMS subscription packages, namely its developed procedures and functions: CONSUME and PROCESS_STAGING. The packages are components responsible for the data integration into the ORMS staging tables, using a methodology of database sharing.

D.1 Store Productivity Subscription Package

Table D.1.1: Store Productivity CONSUME procedure

Name:	CONSUME		
Description:	This procedure will receive an Oracle Object sent by JCAPS and manage the integration logic into ORMS.		
Returns:	N/A		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2(255)	OUT	
O_STATUS	VARCHAR2(1)	OUT	
I_MESSAGE_TYPE	VARCHAR2(15)	IN	
I_MESSAGE_ SUBTYPE	VARCHAR2(15)	IN	
I_MESSAGE	NB_EAI_STORE_PROD	IN	
I_FILENAME	VARCHAR2(255)	IN	
I_STORE	NUMBER(4)	IN	
Development details			
This procedure only will tolerate the following message types: “StoreProdCre”, “StoreProdMod”			
Exception handling:	In case of error, the procedure will send O_Status with “E” and O_Message_Error with the correspondent error message.		

Table D.1.2: Store Productivity PROCESS_STAGING function

Name:	PROCESS_STAGING		
Description:	This function will be launched by the CONSUME procedure and will load into the staging tables the information inside the Oracle Object.		
Returns:	Boolean. False – When the process execute with errors. In this situation, the variable O_Error_Message have to contain the error detail. True – When the process execute without errors.		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2(255)	OUT	
I_MESSAGE_TYPE	VARCHAR2(15)	IN	
I_MESSAGE_ SUBTYPE	VARCHAR2(15)	IN	
I_MESSAGE	NB_EAI_STORE_PROD	IN	
I_FILENAME	VARCHAR2(255)	IN	
I_STORE	NUMBER(4)	IN	
Development details			
Exception handling:	In case of error, all DML instructions at the staging tables will be rolled back and the O_Error_Message will be sent up to the procedure CONSUME.		

D.2 Reception Subscription Package

Table D.2.1: Reception CONSUME procedure

Name:	CONSUME		
Description:	This procedure will receive an Oracle Object sent by JCAPS and manage the integration logic into ORMS.		
Returns:	N/A		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2(255)	OUT	
O_STATUS	VARCHAR2(1)	OUT	
I_MESSAGE_TYPE	VARCHAR2(15)	IN	
I_MESSAGE_ SUBTYPE	VARCHAR2(15)	IN	
I_MESSAGE	NB_EAI_RECEPTION	IN	
Development details			
This procedure only will tolerate one message types: “ReceptionCre”			
Exception handling:	In case of error, the procedure will send O_Status with “E” and O_Message_Error with the correspondent error message.		

Table D.2.2: Reception PROCESS_STAGING function

Name:	PROCESS_STAGING		
Description:	This function will be launched by the CONSUME function and will load the staging tables NB_EAI_RECEPTION_STG and NB_EAI_RECEPTION_DETAIL_STG with the information inside the Oracle Object.		
Returns:	Boolean. False – When the process execute with errors. In this situation, the variable O_Error_Message have to contain the error detail. True – When the process execute without errors.		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2(255)	OUT	
I_MESSAGE_TYPE	VARCHAR2(15)	IN	
I_MESSAGE_ SUBTYPE	VARCHAR2(15)	IN	
I_MESSAGE	NB_EAI_RECEPTION	IN	
Development details			
Exception handling:	In case of error, all DML instructions at the staging tables will be rolled back and the O_Error_Message will be sent up to the procedure CONSUME.		

D.3 RTV Subscription Package

Table D.3.1: RTV CONSUME procedure

Name:	CONSUME		
Description:	This procedure will receive an Oracle Object sent by JCAPS and manage the integration logic into ORMS.		
Returns:	N/A		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2(255)	OUT	
O_STATUS	VARCHAR2(1)	OUT	
I_MESSAGE_TYPE	VARCHAR2(15)	IN	
I_MESSAGE_ SUBTYPE	VARCHAR2(15)	IN	
I_MESSAGE	NB_EAI_RTV	IN	
Development details			
This procedure only will tolerate one message types: “RTVCre”			
Exception handling:	In case of error, the procedure will send O_Status with “E” and O_Message_Error with the correspondent error message.		

Table D.3.2: RTV PROCESS_STAGING function

Name:	PROCESS_STAGING		
Description:	This function will be launched by the CONSUME function and will load the staging tables NB_EAI_RTV_STG and NB_EAI_RTV_DETAIL_STG with the information inside the Oracle Object.		
Returns:	Boolean. False – When the process execute with errors. In this situation, the variable O_Error_Message have to contain the error detail. True – When the process execute without errors.		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2(255)	OUT	
I_MESSAGE_TYPE	VARCHAR2(15)	IN	
I_MESSAGE_ SUBTYPE	VARCHAR2(15)	IN	
I_MESSAGE	NB_EAI_RTV	IN	
Development details			
Exception handling:	In case of error, all DML instructions at the staging tables will be rolled back and the O_Error_Message will be sent up to the procedure CONSUME.		

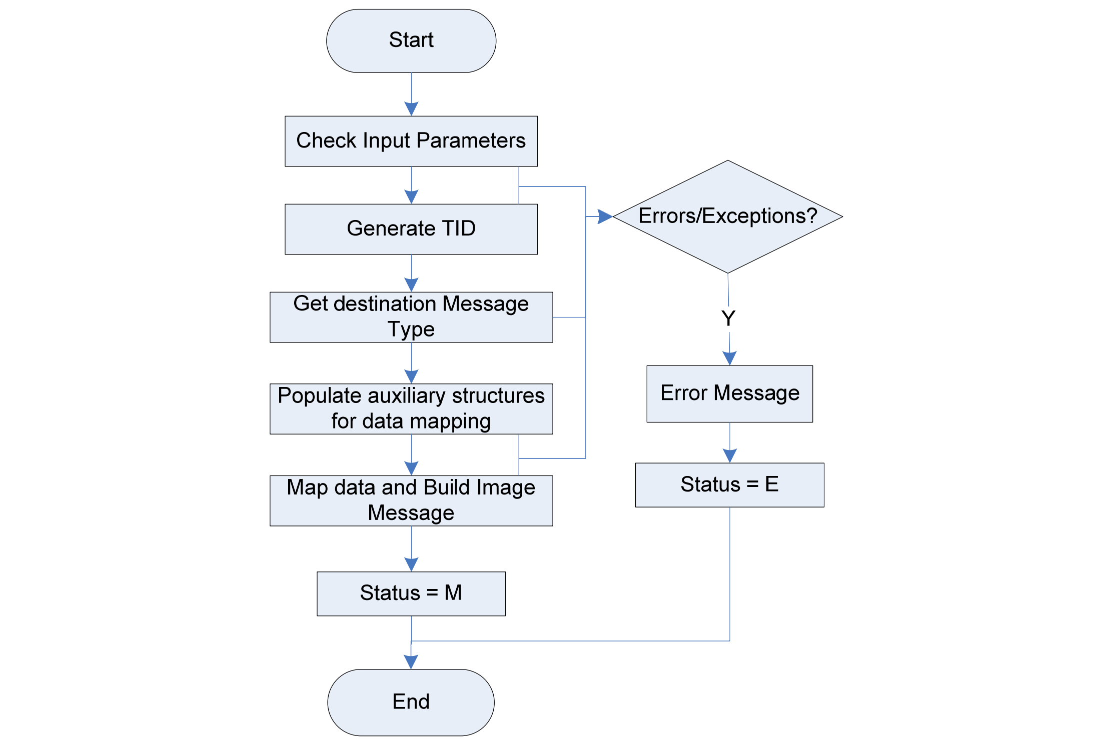
Appendix E: IMAge Mapping Packages

In this appendix is presented a detailed description of the implemented IMAge mapping packages, namely its procedures and functions, for both DAV and ORB entities, referent to the Store to Warehouse Orders and Receptions project. The mapping packages are components responsible for mapping the data, published into the integration BUS by the ORMS application, according to the IMAge input message format.

E.1 MAP Procedure

Table E.1: MAP procedure

Name:	MAP		
Description:	This function is the only exposed procedure and is responsible for the message creation process control flow. It manages the message creation and transformation process.		
Returns:	N/A		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2	OUT	
O_MESSAGE_DST	NB_EAI_IMAGE	OUT	
O_STATUS	VARCHAR2	OUT	
O_TID	VARCHAR2	OUT	
I_MESSAGE_TYPE	VARCHAR2	OUT	
I_MESSAGE_SUBTYPE	VARCHAR2	IN	
I_MESSAGE	ORACLE_OBJECT	IN	
O_ERROR_MESSAGE	VARCHAR2	IN	
Algorithm			


Start
Check Input Parameters
Generate TID
Get destination Message Type
Populate auxiliary structures for data mapping
Map data and Build Image Message
Status = M
End
Errors/Exceptions?
Y
Error Message
Status = E

Development details	
Exception handling:	In case of error, this procedure will return a message, O_ERROR_MESSAGE, with the description of the error and the O_STATUS set to "E".

E.2 GET_TID Function

Table E.2: GET_TID function

Name:	GET_TID		
Description:	This function generates the transaction ID for the message under mapping according to the IMAge Transaction ID specification.		
Returns:	Boolean. False – When the process execute with errors. In this situation, the variable O_error_message have to contain the error detail. True – When the process execute without errors.		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2	IN/OUT	
O_TID	VARCHAR2	OUT	
Algorithm			
IMAge specifies that TID can be generated as a result of 3 concatenated numbers. **TID = LP_TID || JAVA_SYS_TIME_IN_MILLIS || <random natural number between 0 and 9>**			

LP_TID is a constant natural number from 0-9 for every Transaction, is defined at package level.

JAVA_SYS_TIME_IN_MILLIS is computed from the 12 rightmost digits of the Java system time in milliseconds

As an example, taking LP_TID = 2, JAVA_SYS_TIME_IN_MILLIS = 345245674889 and the random number = 5, then the generated TID would be, TID = 23452456748895

Development details	
Exception handling:	In case of error, the function will return false and update the error message accordingly.

E.3 GET_MSG_TYPE Function

Table E.3: GET_MSG_TYPE function

Name:	GET_MSG_TYPE		
Description:	This function sets the message type internally to the package. The message type is used during the transformation process to differentiate sub-messages inside the main family message.		
Returns:	Boolean. False – When the process execute with errors. In this situation, the variable O_error_message have to contain the error detail. True – When the process execute without errors.		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2	IN/OUT	
O_MESSAGE_TYPE	VARCHAR2	OUT	
I_MESSAGE	ORACLE_OBJECT	IN	
Development details			
Exception handling:	In case of error, the function will return false and update the error message accordingly.		

E.4 GET_DESTINATION_MESSAGE Function

Table E.4: GET_DESTINATION_MESSAGE function

Name:	GET_DESTINATION_MESSAGE		
Description:	Function that for a given destination message and application, returns the message ID in the base rule tables and the number of details it contains.		
Returns:	Boolean. False – When the process execute with errors. In this situation, the variable O_error_message have to contain the error detail. True – When the process execute without errors.		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2	OUT	
O_MSG_ID	NUMBER	OUT	
O_DTL_COUNT	NUMBER	OUT	
I_DESTINATION_APPLICATION	VARCHAR2	IN	
I_DESTINATION_MSG	VARCHAR2	IN	
Development details			
Exception handling:	In case of error, the function will return false and update the error message accordingly.		

E.5 FORMAT_FIELDS Function

Table E.5: FORMAT_FIELDS function

Name:	FORMAT_FIELDS		
Description:	This function applies a format to every field of the transformed/mapped message according to definitions stored in the base rules tables.		
Returns:	Boolean. False – When the process execute with errors. In this situation, the variable O_error_message have to contain the error detail. True – When the process execute without errors.		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2	IN/OUT	
O_SRC_VALUE	VARCHAR2	OUT	
I_SRC_VALUE	VARCHAR2	IN	
I_RECORD_ID	NUMBER	IN	
Development details			
Exception handling:	In case of error, the function will return false and update the error message accordingly.		

E.6 FILL_SRC_VALUES Function

Table E.6: FILL_SRC_VALUES function

Name:	FILL_SRC_VALUES		
Description:	This function loads the business data from the Message Oracle Object to a temporary array structure used to support data transformation throughout the mapping package.		
Returns:	Boolean. False – When the process execute with errors. In this situation, the variable O_error_message have to contain the error detail. True – When the process execute without errors.		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2	IN/OUT	
O_SRC_ITEM_DESC	SRC_VALUES	IN/OUT	
I_MESSAGE_TYPE	VARCHAR2	IN	
I_MESSAGE	ORACLE_OBJECT	IN	
Development details			
Exception handling:	In case of error, the function will return false and update the error message accordingly.		

E.7 FILL_EXTINFO_VALUES Function

Table E.7: FILL_EXTINFO_VALUES function

Name:	FILL_EXTINFO_VALUES		
Description:	This function loads the ExtInfo data structure from the Message Oracle Object to a temporary array structure used to support data transformation throughout the mapping package.		
Returns:	Boolean. False – When the process execute with errors. In this situation, the variable O_error_message have to contain the error detail. True – When the process execute without errors.		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2	IN/OUT	
O_EXTINFO	SRC_VALUES_TBL	IN OUT	
I_MESSAGE	ORACLE_OBJECT	IN	
Development details			
Exception handling:	In case of error, the function will return false and update the error message accordingly.		

E.8 MAP_DESTINATION_DETAILS Function

Table E.8: MAP_DESTINATION_DETAILS function

Name:	MAP_DESTINATION_DETAILS		
Description:	This function iterates from message field to message field defined in the base rules tables and maps the field data according the rule type.		
Returns:	Boolean. False – When the process execute with errors. In this situation, the variable O_error_message have to contain the error detail. True – When the process execute without errors.		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2	OUT	
O_MSG	VARCHAR2	IN/OUT	
I_MSG_ID	NUMBER	IN	
I_DTL_COUNT	NUMBER	IN	
I_MESSAGE_TYPE	VARCHAR2	IN	
I_SRC _DATA_DESC	SRC_VALUES	IN	
I_EXTINFO	SRC_VALUES_TBL	IN	
I_TID	VARCHAR2	IN	
Development details			
Exception handling:	In case of error, the function will return false and update the error message accordingly.		

E.9 MAP_DESTINATION_CUSTOM_RULE Function

Table E.9: MAP_DESTINATION_CUSTOM_RULE function

Name:	MAP_DESTINATION_CUSTOM_RULE		
Description:	This function is responsible for all the mapping logic in fields where the defined mapping rule is the rule type = 3.		
Returns:	Boolean. False – When the process execute with errors. In this situation, the variable O_error_message have to contain the error detail. True – When the process execute without errors.		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2	OUT	
O_SRC_VALUE	VARCHAR2	IN/OUT	
I_MESSAGE_TYPE	VARCHAR2	IN	
I_DETAIL	VARCHAR2	IN	
I_SRC_DATA_DESC	VARCHAR2	IN	
Development details			
Exception handling:	In case of error, the function will return false and update the error message accordingly.		

E.10 MAP_DST_CUSTOM_MULTIPLE_RULES Function

Table E.10: MAP_DST_CUSTOM_MULTIPLE_RULES function

Name:	MAP_DST_CUSTOM_MULTIPLE_RULES		
Description:	This function is responsible for all the mapping logic in fields where the defined mapping rule is the rule type = 5. This means that a field is mapped from several input fields and some logic is needed to compute the resulting field.		
Returns:	Boolean. False – When the process execute with errors. In this situation, the variable O_error_message have to contain the error detail. True – When the process execute without errors.		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2	IN/OUT	
O_SRC_VALUE	VARCHAR2	IN/OUT	
I_DETAIL	VARCHAR2	IN	
I_SRC_VALUE_LST	VARCHAR2	IN	
I_SRC_DATA_DESC	SRC_VALUES	IN	
Development details			
Exception handling:	In case of error, the function will return false and update the error message accordingly.		

E.11 MAP_DESTINATION_EXTINFO_RULE Function

Table E.11: MAP_DESTINATION_EXTINFO_RULE function

Name:	MAP_DESTINATION_EXTINFO_RULE		
Description:	This function is responsible for all the mapping logic in fields where the defined mapping rule is the rule type = 4. This means that applies data transformation and custom mapping for message data fields that need information from one field of the ExtInfo data structure.		
Returns:	Boolean. False – When the process execute with errors. In this situation, the variable O_error_message have to contain the error detail. True – When the process execute without errors.		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2	IN/OUT	
O_SRC_VALUE	VARCHAR2	IN/OUT	
I_EXTINFO	SRC_VALUES_TBL	IN	
I_MAIN_LABEL	VARCHAR2	IN	
I_DTL_LABEL	VARCHAR2	IN	

I_MAP_VALUE	VARCHAR2	IN	
I_DETAIL	VARCHAR2	IN	
Development details			
Exception handling:	In case of error, the function will return false and update the error message accordingly.		

E.12 MAP_MULTIPLE_EXTINFO_RULE Function

Table E.12: MAP_MULTIPLE_EXTINFO_RULE function

Name:	MAP_MULTIPLE_EXTINFO_RULE		
Description:	This function is responsible for all the mapping logic in fields where the defined mapping rule is the rule type = 6. This means that applies data transformation and custom mapping for message data fields that need information from multiple fields from the ExtInfo data structure.		
Returns:	Boolean. False – When the process execute with errors. In this situation, the variable O_error_message have to contain the error detail. True – When the process execute without errors.		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2	OUT	
O_SRC_VALUE	VARCHAR2	IN/OUT	
I_SRC_DATA_DESC	SRC_VALUES	IN	
I_SRC_EXTINFO	SRC_VALUES_TBL	IN	
I_DETAIL	VARCHAR2	IN	
I_MAIN_LABEL	VARCHAR2	IN	
I_DTL_LABEL	VARCHAR2	IN	
I_SOURCE_VALUE	VARCHAR2	IN	
Development details			
Exception handling:	In case of error, the function will return false and update the error message accordingly.		

E.13 BUILD_MESSAGE Function

Table E.13: BUILD_MESSAGE function

Name:	BUILD_MESSAGE		
Description:	This function builds the output IMAge message, already transformed and mapped. This assembles the mapped data, the header and details message structure.		
Returns:	Boolean. False – When the process execute with errors. In this situation, the variable O_error_message have to contain the error detail. True – When the process execute without errors.		
Fields			
Name	*Datatype*	*IN/OUT*	*Default*
O_ERROR_MESSAGE	VARCHAR2	IN/OUT	
O_MESSAGE_DST	NB_EAI_IMAGE	OUT	
I_TID	VARCHAR2	IN	
I_MESSAGE_DATA	VARCHAR2	IN	
Development details			
Exception handling:	In case of error, the function will return false and update the error message accordingly.		

Appendix F: Algorithms

In this appendix are presented the developed algorithms which were not included in the body of the report, consisting basically of the algorithms implemented by the Java Collaboration Definition (JCD) used in the developed Publication/Subscription integration flows and the PROCESS_STAGING algorithms of the subscription packages.

F.1 Store Productivity Publisher JCD Algorithm

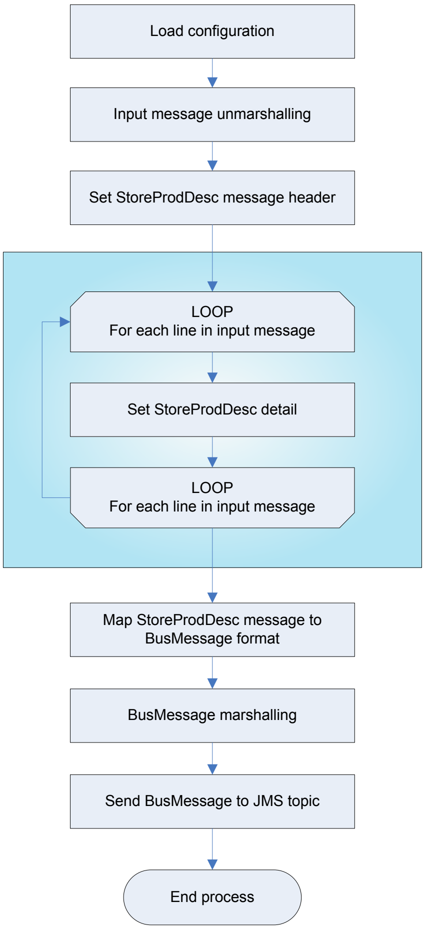

Load configuration
Input message unmarshalling
Set StoreProdDesc message header
LOOP
For each line in input message
Set StoreProdDesc detail
LOOP
For each line in input message
Map StoreProdDesc message to BusMessage format
BusMessage marshalling
Send BusMessage to JMS topic
End process

Figure F.1: Store Productivity publisher JCD algorithm

F.2 FromImageTAFR JCD Algorithm

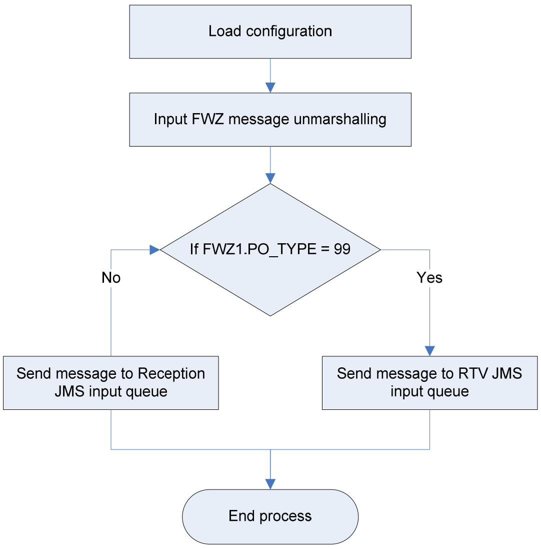

Load configuration
Input FWZ message unmarshalling
If FWZ1.PO_TYPE = 99
No
Yes
Send message to Reception JMS input queue
Send message to RTV JMS input queue
End process

Figure F.2: FromImageTAFR JCD algorithm

F.3 Reception Publisher JCD Algorithm

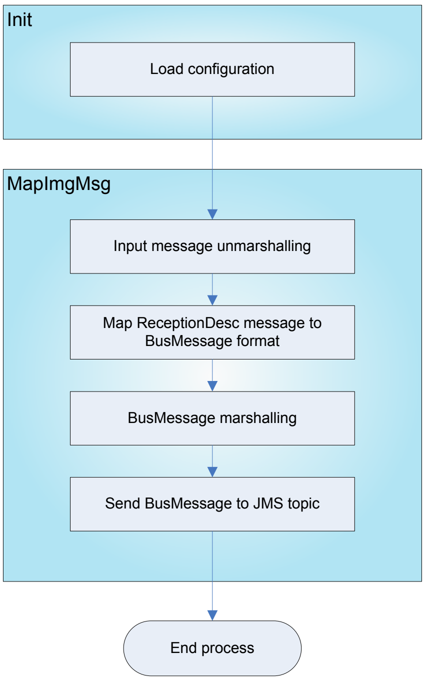

Init
Load configuration
MapImgMsg
Input message unmarshalling
Map ReceptionDesc message to BusMessage format
BusMessage marshalling
Send BusMessage to JMS topic
End process

Figure F.3: Reception publisher JCD algorithm

F.4 RTV Publisher JCD Algorithm

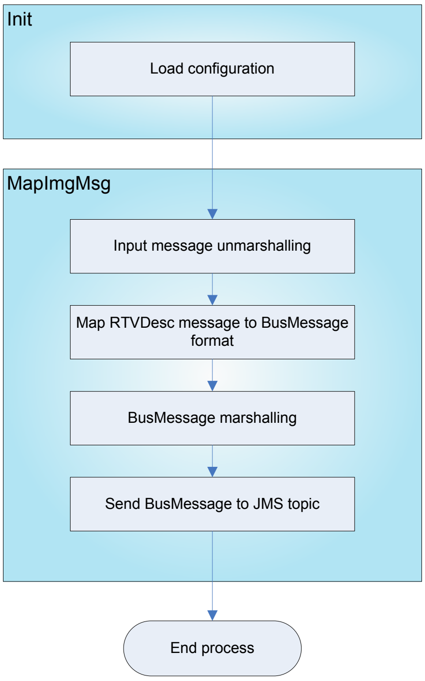

Init
Load configuration
MapImgMsg
Input message unmarshalling
Map RTVDesc message to BusMessage format
BusMessage marshalling
Send BusMessage to JMS topic
End process

Figure F.4: RTV publisher JCD algorithm

F.5 ORMS Subscribers JCD Algorithm

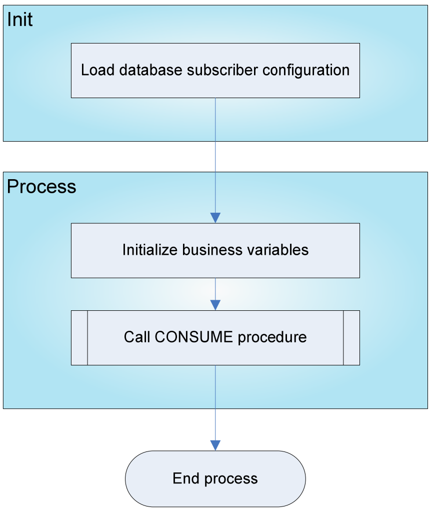

Init
Load database subscriber configuration
Process
Initialize business variables
Call CONSUME procedure
End process

Figure F.5: ORMS subscribers JCD algorithm

F.6 IMAge Subscribers JCD Algorithm

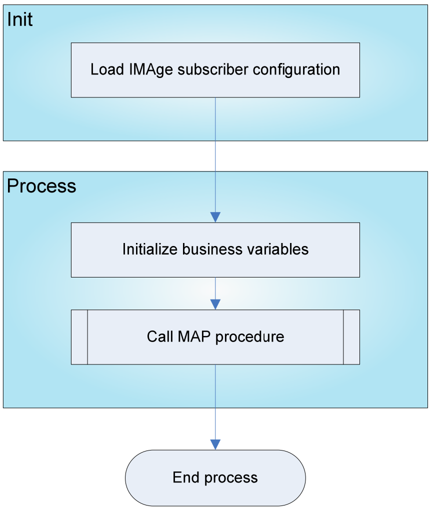

Init
Load IMAge subscriber configuration
Process
Initialize business variables
Call MAP procedure
End process

Figure F.6: IMAge subscribers JCD algorithm

F.7 Store Productivity PROCESS_STAGING Algorithm

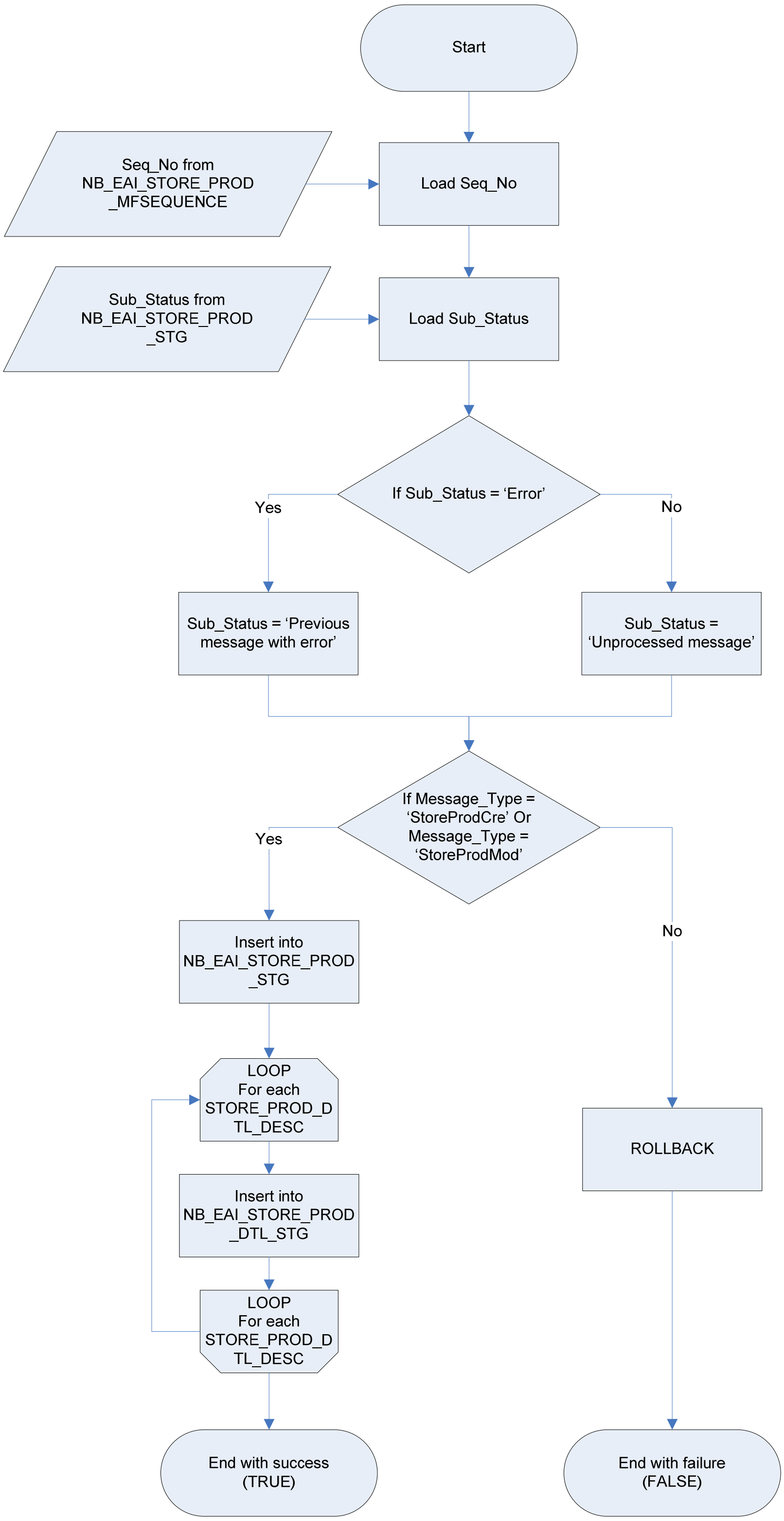

Start
Seq_No from NB_EAI_STORE_PROD _MFSEQUENCE
Load Seq_No
Sub_Status from NB_EAI_STORE_PROD _STG
Load Sub_Status
If Sub_Status = 'Error'
Yes
No
Sub_Status = 'Previous message with error'
Sub_Status = 'Unprocessed message'
If Message_Type = 'StoreProdCre' Or Message_Type = 'StoreProdMod'
Yes
No
Insert into NB_EAI_STORE_PROD _STG
LOOP For each STORE_PROD_D TL_DESC
Insert into NB_EAI_STORE_PROD _DTL_STG
LOOP For each STORE_PROD_D TL_DESC
ROLLBACK
End with success (TRUE)
End with failure (FALSE)

Figure F.7: Store Productivity PROCESS_STAGING algorithm

F.8 Reception PROCESS_STAGING Algorithm

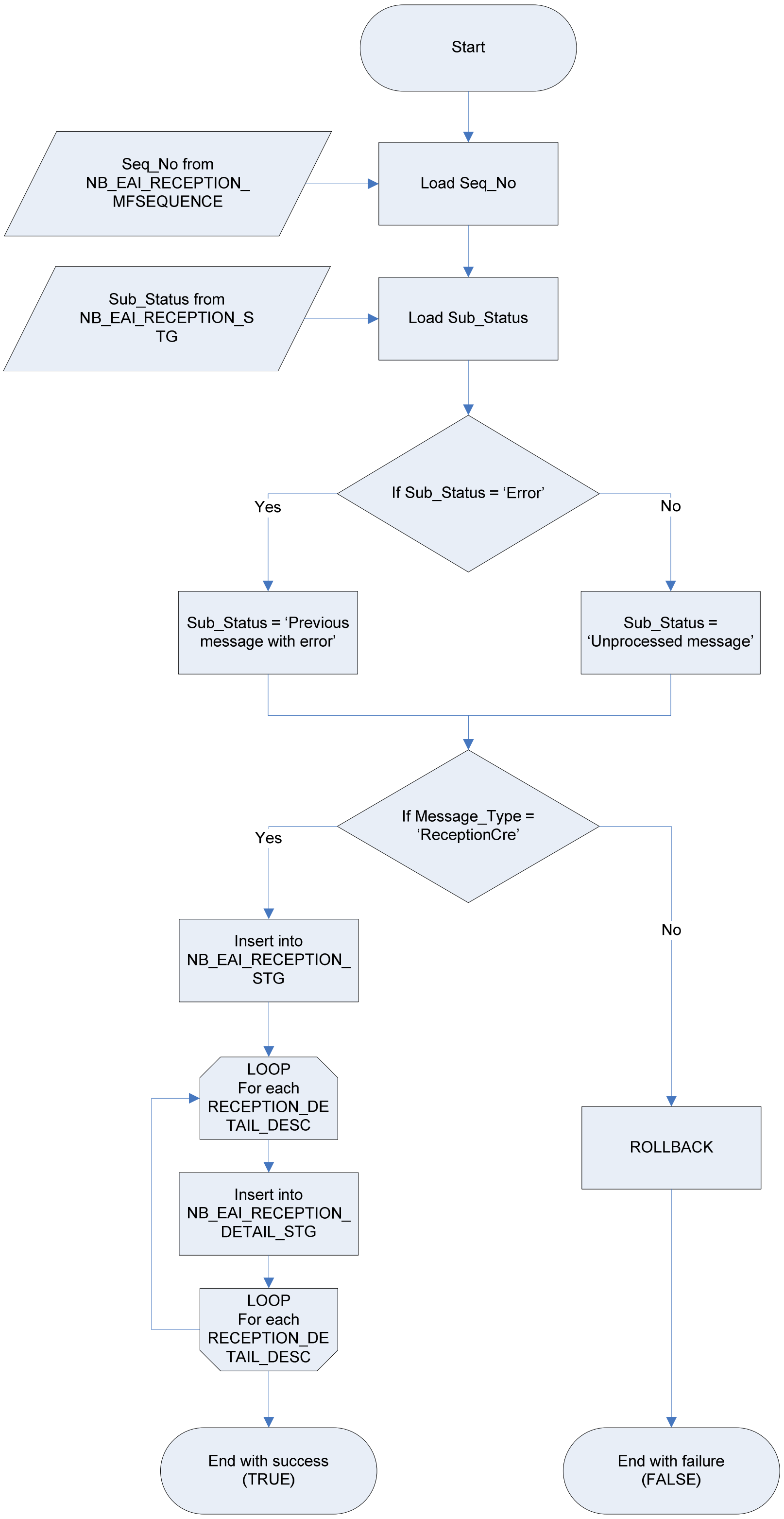

Start
Seq_No from NB_EAI_RECEPTION_ MFSEQUENCE
Load Seq_No
Sub_Status from NB_EAI_RECEPTION_S TG
Load Sub_Status
If Sub_Status = 'Error'
Yes
No
Sub_Status = 'Previous message with error'
Sub_Status = 'Unprocessed message'
If Message_Type = 'ReceptionCre'
Yes
No
Insert into NB_EAI_RECEPTION_ STG
LOOP For each RECEPTION_DE TAIL_DESC
Insert into NB_EAI_RECEPTION_ DETAIL_STG
LOOP For each RECEPTION_DE TAIL_DESC
ROLLBACK
End with success (TRUE)
End with failure (FALSE)

Figure F.8: Reception PROCESS_STAGING algorithm

F.9 RTV PROCESS_STAGING Algorithm

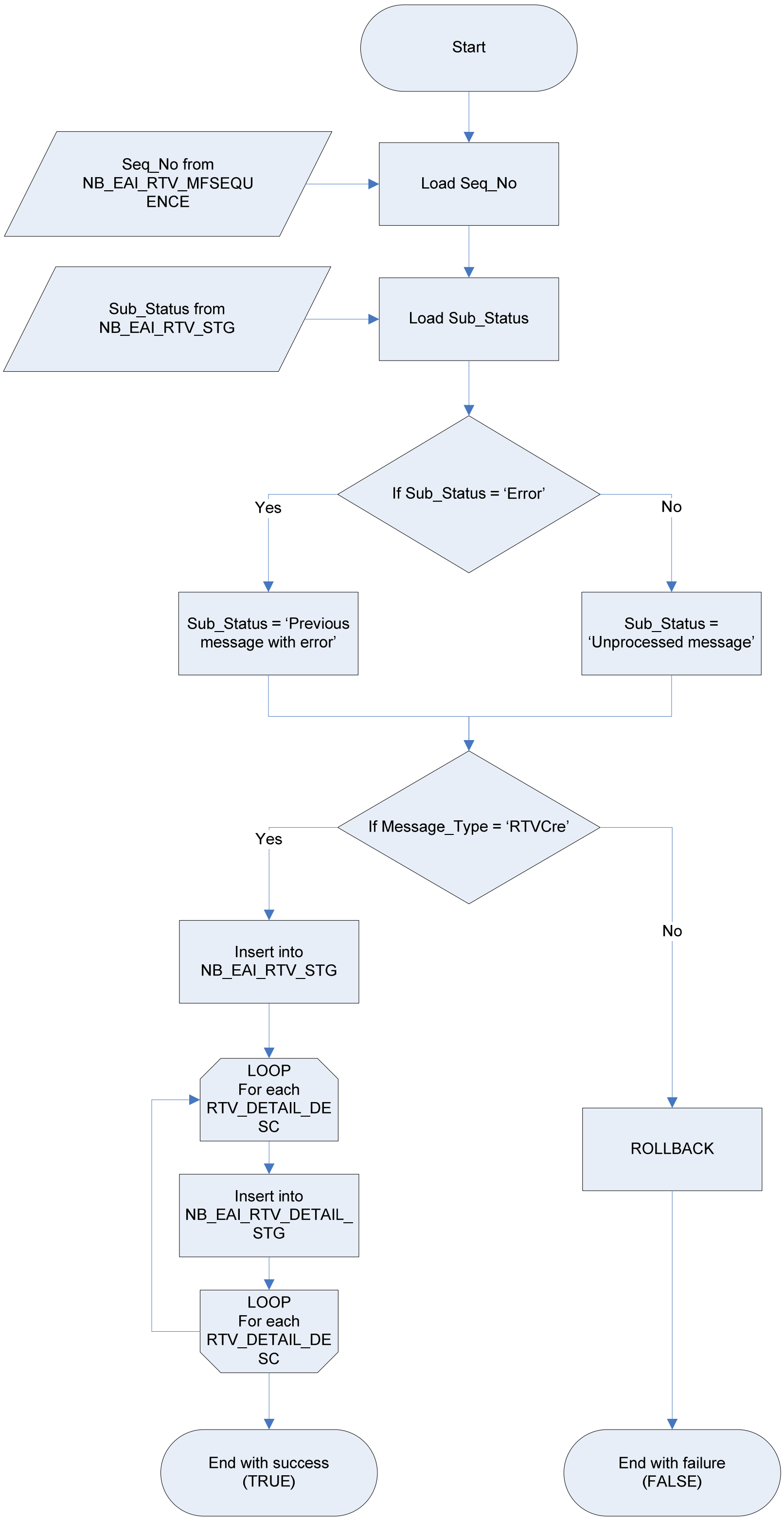

Start
Seq_No from NB_EAI_RTV_MFSEQUENCE
Load Seq_No
Sub_Status from NB_EAI_RTV_STG
Load Sub_Status
If Sub_Status = 'Error'
Yes
No
Sub_Status = 'Previous message with error'
Sub_Status = 'Unprocessed message'
If Message_Type = 'RTVCre'
Yes
No
Insert into NB_EAI_RTV_STG
LOOP For each RTV_DETAIL_DESC
Insert into NB_EAI_RTV_DETAIL_STG
LOOP For each RTV_DETAIL_DESC
ROLLBACK
End with success (TRUE)
End with failure (FALSE)

Figure F.9: RTV PROCESS_STAGING algorithm

Appendix G: Unit Test Documents (UTD's)

This final appendix includes the Unit Test Documents (UTD's) produced for both Store Productivity and Reception/RTV projects.

G.1 Store Productivity UTD

WIPRO
Applying Thought

Unit Test Document

TRD Authors		Developers/Unit Testers			
ID	**Test**	**Tester**	**Date**	**Ok***	**Comments**
101	FTP properties successfully loaded	RSA	05/05/2009	✓	
102	Obtained FTP host file list to be processed	RSA	05/05/2009	✓	
103	All files are processed up to a maximum per round	RSA	05/05/2009	✓	
104	Each of the CSV file contents are mapped into a XML message and sent into a JMS	RSA	05/05/2009	✓	
105	JMS message is mapped correctly for Store Productivity data	RSA	05/05/2009	✓	
106	Message is sent into the integration BUS	RSA	05/05/2009	✓	
107	Message is successfully subscribed	RSA	05/05/2009	✓	
108	Message is inserted into staging tables	RSA	05/05/2009	✓	

TRD Authors		Developers/Unit Testers			
ID	Test	Tester	Date	Ok*	Comments

[*] Use ✓ or X and fill in details in the Comments column in case of errors found

Principles for Software Development			
Principle	Developer	Ok*	Comments
Defensive Programming	RSA RJP	✓	
No Hardcode use	RSA RJP	✓	
Good code readability	RSA RJP	✓	

[*] Use ✓ or X and fill in details in the Comments column in case of nonfulfilment of the Principle

NOTE: The (singular) no fulfilment of any of these principles must be approved by the Project Sponsor. The reason for the no fulfilment must be previously documented, in the project team meeting minute, as well as in the analysis and design documents and in the source code.

Comments *	
TRD Authors *	Developers/Unit Testers **

* Performance expectations. Expected confidence in the unit tests. Etc.

** Performance achieved. Confidence in the unit tests. Aspects to be considered in the integration tests. Etc.

G.2 Reception/RTV UTD

WIPRO
Applying Thought

Unit Test Document

TRD Authors		Developers/Unit Testers			
ID	Test	Tester	Date	Ok*	Comments
101	Received FWZ message from IMAge and sent to TAFR JMS	RSA	22/05/2009	✓	
102	JMS message is mapped correctly for FWZ1 and FWZ2 data	RSA	22/05/2009	✓	
103	If FWZ1.PO_TYPE equals “99” the JMS message is sent to RTV Publisher JMS, otherwise is sent to RECEPTION Publisher JMS	RSA	22/05/2009	✓	
104	Message is sent to the correspondent Subscriber interface	RSA	22/05/2009	✓	
105	Message is inserted into staging tables	RSA	22/05/2009	✓	

* Use ✓ or X and fill in details in the Comments column in case of errors found

Principles for Software Development			
Principle	Developer	Ok*	Comments
Defensive Programming	RSA RJP	✓	
No Hardcode use	RSA RJP	✓	
Good code readability	RSA RJP	✓	

* Use ✓ or X and fill in details in the Comments column in case of nonfulfilment of the Principle

NOTE: The (singular) no fulfilment of any of these principles must be approved by the Project Sponsor. The reason for the no fulfilment must be previously documented, in the project team meeting minute, as well as in the analysis and design documents and in the source code.

Comments *	
TRD Authors *	Developers/Unit Testers **

* Performance expectations. Expected confidence in the unit tests. Etc.

** Performance achieved. Confidence in the unit tests. Aspects to be considered in the integration tests. Etc.

G.3 Store to Warehouse Shipment (DAV) UTD

WIPRO
Applying Thought

Unit Test Document

TRD Authors		Developers/Unit Testers			
ID	**Test**	**Tester**	**Date**	**Ok***	**Comments**
100	PROCESS DATA SENT FOR SUBSCRIPTION INTO THE INTEGRATION BUS				
101	Received WhNotification.Shipment message from the integration BUS, in XML format, corresponding to a "ShipmentCre" message type	RSA	26/06/2009	✓	
102	Message is mapped correctly into its CMF "ShipmentFullDesc"	RSA	26/06/2009	✓	
103	NB_IMGMAP_DAV.MAP procedure is called by the subscriber, returning the transformed message in the IMAge message format	RSA	26/06/2009	✓	
104	NB_IMGMAP_DAV.MAP returns an error status and message if error occurs	RSA	26/06/2009	✓	
105	Message is sent to an IMAge input JMS queue for subscription	RSA	26/06/2009	✓	
200	IMAGE SUBSCRIPTION PACKAGE – MAP PROCEDURE				
201	Input parameters are verified	RSA	26/06/2009	✓	
202	Message TID is successfully generated	RSA	26/06/2009	✓	
203	The incoming message type is correctly determined	RSA	26/06/2009	✓	
204	The destination message ID is properly determined	RSA	26/06/2009	✓	
205	Auxiliary structures of type VARRAY are successfully populated for data mapping	RSA	26/06/2009	✓	
206	ExtInfo auxiliary structure is correctly filled for data mapping	RSA	26/06/2009	✓	

TRD Authors		Developers/Unit Testers			
ID	Test	Tester	Date	Ok*	Comments
207	Each of the fields in the existing message details (DAV2) is successfully mapped according to the defined mapping rules	RSA	26/06/2009	✓	
208	Each of the fields for the message header (DAV1) is successfully mapped according to the defined mapping rules	RSA	26/06/2009	✓	
209	The message header and detail is correctly assembled, returning the final message according to the IMAge message format	RSA	26/06/2009	✓	

[*] Use ✓ or X and fill in details in the Comments column in case of errors found

Principles for Software Development			
Principle	Developer	Ok*	Comments
Defensive Programming	RSA	✓	
No Hardcode use	RSA	✓	
Good code readability	RSA	✓	

[*] Use ✓ or X and fill in details in the Comments column in case of nonfulfilment of the Principle

NOTE: The (singular) no fulfilment of any of these principles must be approved by the Project Sponsor. The reason for the no fulfilment must be previously documented, in the project team meeting minute, as well as in the analysis and design documents and in the source code.

Comments *	
TRD Authors *	Developers/Unit Testers **

* Performance expectations. Expected confidence in the unit tests. Etc.
** Performance achieved. Confidence in the unit tests. Aspects to be considered in the integration tests. Etc.

G.4 Store to Warehouse OrderResponse (ORB) UTD

Unit Test Document

TRD Authors		Developers/Unit Testers			
ID	**Test**	**Tester**	**Date**	**Ok***	**Comments**
100	PROCESS DATA SENT FOR SUBSCRIPTION INTO THE INTEGRATION BUS				
101	Received WhNotification.OrderResponse message from the integration BUS, in XML format, corresponding to a "OrderResponseCre" message type	RSA	26/06/2009	✓	
102	Message is mapped correctly into its CMF "OrderResponseFullDesc"	RSA	26/06/2009	✓	
103	NB_IMGMAP_ORB.MAP procedure is called by the subscriber, returning the transformed message in the IMAge message format	RSA	26/06/2009	✓	
104	NB_IMGMAP_ORB.MAP returns an error status and message if error occurs	RSA	26/06/2009	✓	
105	Message is sent to an IMAge input JMS queue for subscription	RSA	26/06/2009	✓	
200	IMAGE SUBSCRIPTION PACKAGE – MAP PROCEDURE				
201	Input parameters are verified	RSA	26/06/2009	✓	
202	Message TID is successfully generated	RSA	26/06/2009	✓	
203	The incoming message type is correctly determined	RSA	26/06/2009	✓	
204	The destination message ID is properly determined	RSA	26/06/2009	✓	
205	Auxiliary structures of type VARRAY are successfully populated for data mapping	RSA	26/06/2009	✓	
206	ExtInfo auxiliary structure is correctly filled for data mapping	RSA	26/06/2009	✓	
207	Each of the fields in the existing message details (ORB2) is successfully mapped according to the defined mapping rules	RSA	26/06/2009	✓	
208	Each of the fields for the message header (ORB1) is successfully mapped according to the defined mapping rules	RSA	26/06/2009	✓	
209	The message header and detail is correctly assembled, returning the final message according to the IMAge message format	RSA	26/06/2009	✓	

TRD Authors		Developers/Unit Testers			
ID	Test	Tester	Date	Ok*	Comments

* Use ✓ or X and fill in details in the Comments column in case of errors found

Principles for Software Development			
Principle	Developer	Ok*	Comments
Defensive Programming	RSA	✓	
No Hardcode use	RSA	✓	
Good code readability	RSA	✓	

* Use ✓ or X and fill in details in the Comments column in case of nonfulfilment of the Principle

NOTE: The (singular) no fulfilment of any of these principles must be approved by the Project Sponsor. The reason for the no fulfilment must be previously documented, in the project team meeting minute, as well as in the analysis and design documents and in the source code.

Comments *	
TRD Authors *	Developers/Unit Testers **

* Performance expectations. Expected confidence in the unit tests. Etc.
** Performance achieved. Confidence in the unit tests. Aspects to be considered in the integration tests. Etc.