Design and Instantiation of Reference Architecture of Pluggable Service Platform in E-Commerce

Master Thesis University of Twente

Mohammad Anggasta Paramartha

8/28/14

S1341243

Design and Instantiation of Reference Architecture of Pluggable Service Platform in E-Commerce

Master Thesis

Final Version

Enschede, 28 August 2014

Author:

Name: Mohammad Anggasta Paramartha

Program: Master of Industrial Engineering and Management

Institution: University of Twente

Email: mohammadanggastaparamartha@student.utwente.nl

Graduation Committee:

University of Twente

First Supervisor: Dr. Maria-Eugenia Iacob

Second Supervisor: Prof. Jos van Hillegersberg

External Committee: Fabian M. Aulkemeier

Management Summary

In the beginning of e-commerce era, retailers introduced online channel mainly by adopting vertically integrated e-commerce solutions to gain control of all e-commerce functionalities. However, it is considered to be nearly impossible for a single platform to be the industry-leading expert in all the functionalities. If the functionalities are tightly coupled to the platform, the options for the retailer become severely limited. In addition, retailers began to realize that enterprise agility could be compromised. In order to maintain agility, it is advised to have a lightweight, modular, and flexible architecture with small core functionalities. Other functionalities then can be plugged-in to the platform through third party providers. With this approach, retailers gained the ability to customize the platform to meet their specific needs.

Companies eventually started to shift from the monolithic and vertically integrated systems towards a collaborative network of partners within the value chain. In this type of ecosystem with diverse software systems and technology of the network partners, integration and interoperability become critical factors to enable information exchange and seamless coordination among the partners. Recent development in Information Technology field has also led to proliferation of new technologies such as cloud computing, mobile and social media. As a consequence of this situation, retailers are facing new integration challenges. Failure to cope with these challenges will eventually result in decrease in enterprise agility, which ultimately will have negative impact on overall business performance.

Objective

Thus, this research aims to design and develop a reference architecture of a novel pluggable service platform in e-commerce. This platform should be able to support seamless integration and coordination of e-commerce supply chain partners' diverse applications and services, which could include aforementioned emerging technologies (cloud computing, mobile, SaaS, social media)

Methodology

In this research, we use a research methodology called Design Science Research Methodology (DSRM). DSRM is about solving problems by introducing artifacts in a context. The artifact that we propose is a reference architecture for pluggable service platform in e-commerce. This methodology comprises of six stages which correspond with each chapters of this report. First, state-of-the art analysis will be performed to understand the currently available e-commerce web shop platform and integration platform solutions in the market. Based on the findings of the market analysis and motivated by literature study on service-orientation paradigm, the reference architecture will be designed. This proposed design will be then instantiated by means of a prototype for a specific use-case in e-commerce to demonstrate the feasibility of our architecture design. Finally, the platform design will be evaluated.

Key Findings

The main output of this research is the reference architecture design of the pluggable service platform. Around this output, some key findings from each chapter in this report could be concluded:

- Latest development and trend of e-commerce web shop platform solutions have been investigated in chapter 2. The main finding is the list of functionalities that are commonly provided in a basic e-commerce packages. These functionalities are used as an input for the e-tailer part in the complete reference architecture
- State-of-the art of integration platform solutions has been elaborated in chapter 3. The main result from this chapter is the 'Collaborative Service & Process Framework' component whose functionalities are derived from SOA Governance and API Management technologies.
- Chapter 4 contains the complete reference architecture design, which adhere to service oriented design principle and has been constructed with TOGAF framework and Archimate modeling language. The highlight of this chapter is the 'Collaborative Data Management' component, which proposes the creation of a canonical data model, equipped with schema mapper function, as a reference of data model for external services
- Prototype realization in chapter 5 demonstrates the applicability of our platform design to be applied to solve real e-commerce case. This also shows the suitability of the cloud-based integration platform solution in the market to fulfill the goal of this research. Through the evaluation phase, we realized that there are no well-defined metrics to assess pluggability in quantitative ways. We then assess the extent to which our platform design has supported the goal of this research by means of agility. Agility was proved in this research to be suitable as a surrogate measure to pluggability.

Preface & Acknowledgement

This thesis report is the culmination of my journey in the Netherlands to pursue a master degree. Time flies. Two years feel just like days. All the memories here, even the first day I stepped my feet in the Netherlands, still remain vivid in my mind. I remember on January 2014, I was still looking for a graduation project. Then, Maria offered me this project which I eventually accepted because the topic was very interesting for me. Six months later, I successfully completed this master assignment.

The past six months has been a really priceless experience for the development of myself both personally and professionally. I have learned a lot of new things within e-commerce and integration world, had inspiring discussions with my supervisors, met industry experts during Catelog consortium meetings, and I had the chance to attend the API Strategy Conference in Amsterdam where I gained a lot of knowledge on the latest developments in API world.

This project would not have been successfully completed without the support, advices and presence of certain entities and people. First of all and the most important, I would like to express my sincere gratitude to my God Allah SWT for the blessing and guidance throughout my life and my study phase.

I would like to extend my gratitude for the Indonesian Ministry of Communications and Information Technology for the scholarship opportunity given to me. I promise to use the knowledge and experience that I obtained here to the fullest for the development of Indonesia, particularly in ICT sector.

I would also like to express my deepest gratitude to my thesis supervisors: Dr. Maria-Eugenia Iacob as my first supervisor and Prof. Jos van Hillegersberg as my second supervisor, for their trust in me, never-ending support, insightful guidance, and all the life & career advices. Thank you for the opportunity to be a part of this Catelog project and the invitations to join the consortium meeting. I would like to extend this appreciation to Fabian M. Aulkemeier, my daily supervisor who has given me continuous support and assistant, especially with the technical parts, for backing me up during many thesis meetings, and for the patience and willingness to read and correct my thesis report. It has been an honor for me to have the chance to work with you all.

Last, I would like to deliver special thanks for my family in Indonesia, especially my mother who always motivates me and prays for me every single day; my girlfriend in Indonesia who is patiently waiting for me to come back home; my family in PPI Enschede, IMEA and PPI Belanda for the togetherness, for sharing joys and pains, for making my days here more colorful and feels like home; KBRI Den Haag (Indonesia Embassy in the Netherlands) for the assistance especially with the administrative matters; and finally all my Dutch and international friends & colleagues who have given unique experiences during my study here. I am going to miss you all. Now it is the time for me to start a new life, a new journey.

Mohammad Anggasta Paramartha

Enschede, August 2014

Table of Contents

Table of Figures

Table of Tables

1. Introduction

This introduction chapter outlines the general structure of this thesis report and describes the research that has been carried out. In the first section, the background and motivation behind this research are explained. Section 1.2 discusses the problems to be addressed by this research and accordingly, the research goal. The subsequent section contains a formulation of the main research question and sub-questions derived from the main question. Section 1.4 describes the research method and approach. Last, section 1.5 lists the structure of this report.

1.1 Background and Motivation

The vast developments of internet applications and infrastructures recently has led to shifts in how people make purchase of goods. Traditionally, customers need to first travel to reach the physical store, spend some time wandering inside the store, before finally purchase items that they want. If they can't find the item that they want, they might need to move to other stores. The whole process could take a lot of time and energy only to travel from one place to another. Nowadays, due to increased adoption of online shopping channel, customers could easily open websites of any retailer to conveniently buy any items that they want, without having to leave their place. The term "e-commerce" or electronic commerce emerged for this new channel of economic transactions.

In the past several years, there has been dramatic and continuous increase of the Business to Customer (B2C) online sales all over the globe which has reached 20% of global annual growth, in contrast with the declining traditional offline channel (Ystats.com, 2013). Europe had overtook USA as the largest B2C e-commerce market since 2010. According to E-commerce Europe (2013), the size of European B2C e-commerce grew by 19% in 2012, reaching € 311.2 Billions of turnover. Specifically in the Netherlands, Thuiswinkel.org research shows that Dutch online consumer sales grew 10 percent in 2012 compared with 2011, reaching to almost € 10 Billions of turnover (Ecommerce Europe, 2013).

The figures above indicate a bright future of e-commerce industry, globally as well as regionally. In line with this promising future, customer expectations are also growing in terms of shopping experience and order fulfillment. Buyers favor shops that can provide easy access to product information and transaction as well as short delivery times at a low price. Research shows failure to live up to order fulfillment promises by e-tailers (electronic retailers) can be detrimental to online sales, with out-of-stocks strongly correlating negatively with a consumer's loyalty to a web shop (Rao, Griffis, & Goldsby, 2011).

A well-structured logistics program also can create substantial value-added and positively affect the bottom-line of the process (Bernon, Rossi, & Cullen, 2011; Genchev, Richey, & Gabler, 2011). Early entrants such as etoys.com went bankrupt because of lack of attention to supply chain aspects (Pyke, Johnson, & Desmond, 2001). Thus, out of all factors that are considered essential to the success of e-commerce platform, the supply-chain performance from suppliers down to the doorstep a customer has been increasingly recognized as the main success factor. As a leading country in logistics, the Dutch e-commerce community could take an important share from the European Union online sales.

In the beginning of e-commerce era, retailers introduced online channel mainly by adopting vertically integrated e-commerce solutions to gain control of all e-commerce functionalities, ranging from suppliers side to distributors side. A basic e-commerce platform package became congested with a huge number of built-in functionalities. This approach seems to be beneficial at first, for instance in terms of time to market and supply control. However, retailers then began to realize that enterprise agility can be compromised. Agility plays an important role to maintain competitive advantage in the dynamic and ever-changing e-commerce market. In order to achieve agility, a better approach for retailers would be to focus on certain core e-commerce capabilities and then create a partner ecosystem around them as a mean to fulfill other capabilities that might be needed. From technical point of view of e-commerce platform, this also means that it is advised to have a lightweight, modular, and flexible platform architecture with few core e-commerce functionalities which then can be extended by additional services from third party providers.

Companies eventually started to shift from the monolithic and vertically integrated systems towards a collaborative network of partners within the value chain. In this type of ecosystem with diverse software systems and technology of the network partners, integration and interoperability become critical factors to enable information exchange and seamless coordination among the partners (Mulesoft, 2013). Companies should be able to find ways to ensure seamless integration in the level of data, application and business process, both within and across-enterprise. Within this specific aspect, an ICT architecture for information systems that can support collaborative service-based process will be designed in this research. The architecture will be the underlying design of a pluggable service platform for e-commerce, which can be used by business partners within e-commerce value chain to achieve better and more seamless integration and collaboration.

1.2 Research problems & goals

As can be inferred from the previous section, tackling integration and interoperability challenges is imperative in order to fully reap the benefits of e-commerce. Integrating disparate information systems of supply chain partners becomes the requisite to achieve successful collaboration. Traditionally, Business-to-Business (B2B) integration tasks were mostly accomplished by implementing Electronic Data Interchange (EDI). Up to now, EDI is still the most dominant technology that companies use to exchange information electronically. Nevertheless, setting up EDI is perceived to be expensive, complex and time-consuming. EDI is also unsuitable for a typical B2B integration over internet (Samtani, 2002).

Furthermore, recent development in Information Technology field has led to proliferation of new technologies such as cloud computing, mobile and social media. An increasing adoption of cloud computing technology could be observed, with more and more companies moving their applications and services towards externally hosted and managed ICT platforms and applications. According to a market report by Research In Action (2012), 78% of companies under its study have adopted e-commerce cloud services, making it the most widely used type of cloud service. Nevertheless, some enterprise applications, especially legacy and back-office systems, have to remain on-premise due to security and confidentiality issues. These legacy systems will then continue to exist but new types of applications like SaaS (Software as a Service) will be incorporated into the enterprise application landscape.

As a consequence of this situation, retailers are facing more challenging integration issues. The legacy systems are not designed to interoperate with the new technologies. Similarly, while traditional on-premise integration platforms support complex integration with legacy systems, they are not specifically architected in coping with integration scenarios which involve cloud applications due to different characteristics with their on-premise counterparts. The problem bundle in Figure 1 depicts the aforementioned situations and their possible consequences to integration and interoperability. Failure to cope with these challenges will eventually result in decrease in enterprise agility, which ultimately will have negative impact on overall business performance.

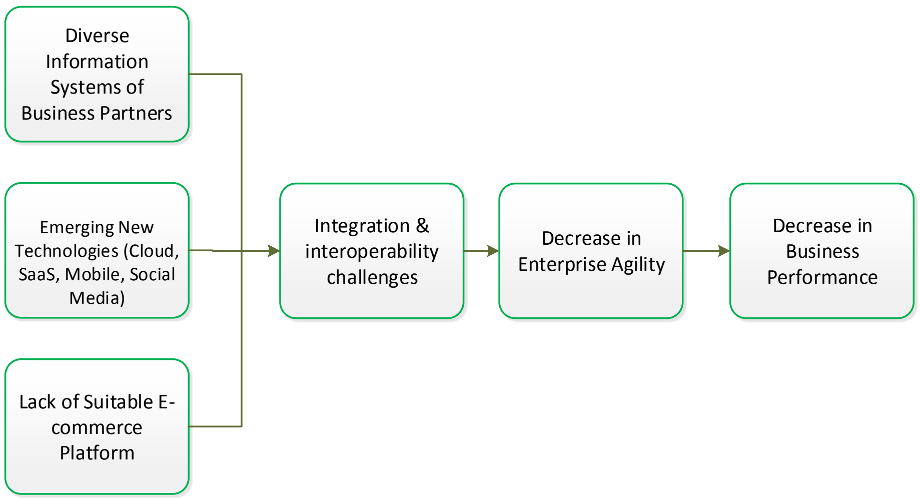

Diverse Information Systems of Business Partners
Emerging New Technologies (Cloud, SaaS, Mobile, Social Media)
Lack of Suitable E-commerce Platform
Integration & interoperability challenges
Decrease in Enterprise Agility
Decrease in Business Performance

Figure 1 Research problem bundle

Through this research, we aim to find solutions for the issues discussed above. In a bigger scale, the primary goal of the project is to enable Dutch retailers with online channels to increase their market share and revenues through best-in-class logistics and fulfilment. Particularly in this research, we intend to design a more suitable type of e-commerce platform with regards to integration and coordination requirements. The platform will be designed in the form of a reference architecture.

It was stated by W3C (2004) that designing a reference architecture could serve as an initial step in the process of creating a software architecture for a specific IS. A reference architecture contains a generic structure of system elements and their functions and interfaces within a particular domain. To construct the architecture, one needs to understand common aspects in the IS configuration and the function in that domain. In the domain of e-commerce, we will explore these following aspects:

- Common architectural components that currently available E-commerce platform and integration platform solutions cover and lack of
- Actors, applications and IT infrastructures in typical E-commerce value chain
- Architecture design principle, framework and modeling language to adhere
- Functional and non-functional requirements of the platform
- Suitable e-commerce use case and tools to instantiate the architecture as prototype

Thus, the main research objective of this project is:
To develop a reference architecture of pluggable e-commerce platform which supports seamless integration and coordination of e-commerce supply chain partners' diverse applications and services

The reference architecture will be validated through instantiation of the design as a prototype which will demonstrate the applicability of our design in practice. We will choose a specific case in e-commerce delivery process that might be interesting to study.

By achieving the research objective, this research could bring contribution to both theory and practice. This research will bring considerably advancement to the development of scientific knowledge on the interface of logistics, marketing and ICT in e-commerce. This research also demonstrates applicability of our agile software architecture to solve real industry problems and accordingly, reduce gaps between theory and practice. The architecture design itself also promotes extensibility by pluggable software services, which is relatively new in theoretical context.

We also expect that this project will bring substantial implication to practice. This research will contribute to the further development of the e-commerce sector with regards to improvement of opportunity in logistics and stock management. We provide latest insights of e-commerce best practices in through market analysis and benchmark study. Ultimately, we expect that this project can contribute considerably to the ambition set forward to create new innovations and support Dutch retailers to increase their revenues in this e-commerce sector.

1.3 Research questions

This research is conducted to achieve previously stated objective by answering the following **main Research Question:**
What reference architecture can best serve as the foundation for a pluggable e-commerce platform which supports seamless integration and coordination of e-commerce supply chain partners' application and services?

From the main research question, we derived the following **sub-research questions:**

Sub-RQ1: What is the current e-commerce platform solutions landscape?
- What is the standard architecture of current e-commerce web shop platforms?
- What features/system components are generally provided? What features that might be necessary but the current platforms typically lack of?
- What issues are associated to the platforms with regards to integration and coordination?

Sub-RQ 2: How is the current integration platform solutions landscape?
- What is the role of integration in E-commerce domain?
- What is the standard architecture of current integration platforms?
- What features/system components are generally provided? What features that might be necessary but the current platforms typically lack of?

Sub-RQ3: How to design the reference architecture of pluggable E-commerce platform which support seamless integration and coordination?

- What architecture design principle and architecture modeling language to adhere?
- How should Business, Information System, and Technology domains of the platform be constructed?
- What components should be included in the architecture?

Sub-RQ4: How to implement and evaluate the reference architecture?
- What approach and tools to use to instantiate the architecture as prototype?

- Which parts of the e-commerce process to be selected as the case study of the prototype?
- How to evaluate the design of the architecture?

1.4 Research methodology

In this research, a research methodology called Design Science Research Methodology (DSRM) was employed. DSRM is about solving problems by introducing artifacts in a context. The artifact that we propose is a system architecture for pluggable e-commerce platform. Research phases that have to be carried out in DSRM are (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007):

(1) Problem definition & analysis (evaluation of current practice)
(2) Defining objectives of a solution (what would a better artifact accomplish?)
(3) Artifact design & development
(4) Artifact demonstration (finding a suitable context then use the artifact to solve problems)
(5) Artifact evaluation (observing how effective it is in solving problem)
(6) Communication.

After this point usually the process iterates back to step (2) or (3). Following this DSRM method, we first investigate the market to gain insight on currently available e-commerce platform solutions and architecture (step 1 in DSRM). Based on the findings of this market analysis, we will identify issues associated with the platforms with respect to integration and coordination, which provides motivation for the need of a new platform. Common technology used, architecture components and functionality gaps will be acknowledged as well. Step (1) will be covered by chapter 1 and 2 in this report.

In the next step (2), we will propose requirements and architecture components that need to be incorporated into the platform design based on literature study. This phase is necessary to accommodate the inadequacy of solutions in the market in achieving our project goals. We will carry out literature study in the topics of e-commerce services, service platform, integration and coordination in e-commerce logistic, and other relevant topics.

Subsequent step (3) in DSRM is about artifact design and development. Based on the findings of chapter 1, 2 and 3, we will construct an architecture design of our platform in this chapter 4. We will first study what architectural design principle to implement and accordingly, its design specification and requirement. As the basis of process model in our reference architecture, we will also investigate existing reference models for e-tailing to understand both business and IS domain in the entire process of order-to-delivery e-commerce.

Chapter 5 in this report corresponds with Step (4) which deals with artifact demonstration, and Step (5) which is about artifact evaluation. We will find a suitable context within e-commerce order fulfilment process to demonstrate the feasibility of our architecture design by means of a prototype. We will choose a scenario that might be interesting for our project partners and has not been covered by majority of e-commerce solution nowadays. The case selection will be based on consideration from literatures and insight from discussions with project partners. In addition, we will select a tool to instantiate the architecture design into prototype. By constructing the prototype using real world problem and real world services, the prototype is automatically validated.

The last section is Chapter (6) which correspond to Step (6) in DSRM research phase. We conclude this thesis with discussion and limitations then point out recommendations for future researches. This thesis as well as conference papers generated from it serve as means of communication to public.

Figure 2 below reflects overview of our research approach. The numbers in the box refer to chapters addressing the concepts.

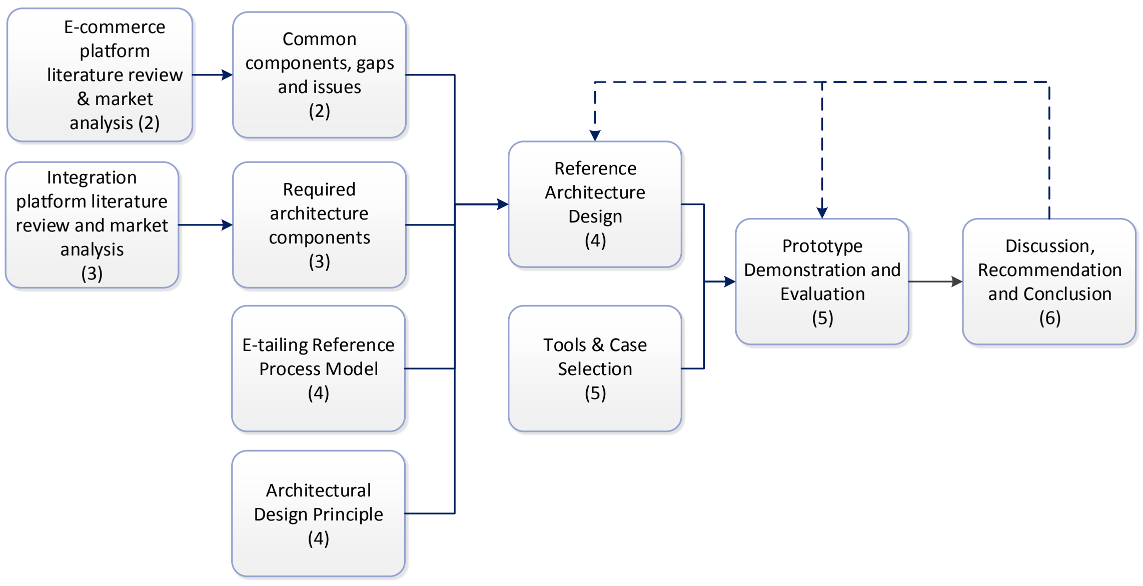

E-commerce platform literature review & market analysis (2)
Common components, gaps and issues (2)
Integration platform literature review and market analysis (3)
Required architecture components (3)
E-tailing Reference Process Model (4)
Architectural Design Principle (4)
Reference Architecture Design (4)
Tools & Case Selection (5)
Prototype Demonstration and Evaluation (5)
Discussion, Recommendation and Conclusion (6)

Figure 2 Research approach and chapters

1.5 Report structure

The remainder of this report is structured in correspondence with each research questions:

Sub-RQ1 is answered by Chapter 2: Literature study and state-of-the-art analysis of E-commerce web shop platform solution

Sub-RQ2 is answered by Chapter 3: Literature study of integration in E-commerce field and state-of-the-art analysis of cloud-based integration platform solutions

Sub-RQ3 is answered by Chapter 4: Reference architecture design and development

Sub-RQ4 is answered by Chapter 5: Case selection and platform prototype

Chapter 6: Discussion, contributions, conclusions, recommendations for further work

2. E-Commerce

Rapid development of information technology and internet technology has led to a shift in how people do business. A new type of industry called electronic-commerce or e-commerce emerged as a result of the shift. E-commerce generally refers to any type of commercial transactions of goods and services performed over the internet (Maamar, 2003). E-commerce put new demands on people, process and technology involved in supporting this electronic transaction.

This chapter elaborates about e-commerce industry and the latest e-commerce web shop platform solutions. First, e-commerce industry and its relation with e-tailing/electronic retail is explained. Then, a brief overview about the e-commerce web shop platform landscape is described in chapter 2.2 and common features of the platform is investigated in section 2.3. Afterwards, the current state-of-the-art of e-commerce platform market is investigated in section 2.4. Deficiencies and issues of the current e-commerce platforms are discussed in chapter 2.5 and conclusions are drawn upon.

2.1 E-Commerce and E-tailing Industry

A large and growing body of literature has investigated the topic of e-commerce. Much of the literatures within this e-commerce domain have paid particular attention to give a clear definition of what e-commerce term actually is because the definitions given by various sources could differ significantly and the term is frequently misused to represent different meanings.

One definition of electronic commerce is the use of computer networks to conduct business (buying and selling of goods and services) using integrated set of electronic tools with one's suppliers, customers, and/or competitors (Hayashi, 1996) with the purpose to streamline business processes and reduce cycle time (Benesko, 1994). Besides that, Weill & Vitale (2013) stated that E-commerce describes the roles and correlation of customers, clients, co-operators and suppliers of an enterprise. Based on those descriptions, we could indicate that e-commerce is closely related to flow of products, information, and capitals, and the main benefits of every partners. Electronic Retailing, or E-tailing, is a subset of E-commerce industry and one of the earliest applications of e-commerce. To put it simply, e-tailing refers to selling of retail goods, such as books, clothing or merchandise, over the internet (Huang & City, 2011).

By performing sales transaction over the internet, both retailers and consumers could benefit in several aspects (Endo, Yang, & Park, 2012; Eroglu, Machleit, & Davis, 2001). Benefits for e-retailers compared to traditional retail are mainly in cost saving because of less paper work, lower customer acquisition costs and significantly lower initial investment as they don’t have to build physical stores and hire store staffs. Retailers also have the opportunity to reach more customers due to no geographical restrictions.

Customers can benefit from a lot more available product options as they could easily browse on the internet from one store to the other to find information for goods that they want and make comparison between the retailers. This process saves a lot of time and also brings a great deal of convenience for costumers as they only need to sit in front of their computer to perform transaction. Customers could also benefit from reduction in information searching costs to obtain better product by looking at online rating/review of a specific product provided by other customers. As a result, customers can make better decision before purchasing a product and get best possible prices or delivery time of the same good.

2.2 E-commerce Web Shop platform

In the early days of e-commerce, e-commerce websites were implemented mostly for publishing product catalog over the internet. Retailers created online presence solely through standard and simple websites. Customers then can find information of available products in the website but they still had to visit the physical stores to make purchases. Only few big retailers provided full range of e-transaction functionalities from displaying product catalog to payment and shipping. Most of the e-commerce websites were created as tailor-made solutions because there were no affordable and mature e-commerce solution back then. Accordingly, companies had to adjust their e-commerce website according to their resources and needs. Development time were long and companies needed their own IT team to manage and maintain the website.

In recent years, e-commerce platform solution landscape has evolved from the custom-made to pre-packaged web shop solutions. A pre-packaged web shop solution provides basic e-commerce functionalities such as shopping cart, product catalogue management, marketing tools, or payment. Besides the basic functionalities, usually it is also possible to configure and customize the web shop through integration with 3rd party services to meet specific needs of online merchants. By implementing the pre-packaged solution, it becomes easier for business owner to set up and launch their online store, resulting in faster time to market. (Chu, Leung, Hui, & Cheung, 2007)

A full-service e-commerce solution provides support for the entire E-commerce transactions and collaboration among enterprises, which including series of activities, from raw materials, procurement, resource management, product display, ordering to production, storage, transportation, electronic payment and customer management. In addition, a full-service E-commerce platform also enables management of internal business activities. As a consequence, it should be able to integrate wide range of enterprise systems like Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), or Supply Chain Management (SCM). (Zhonghua & Erfeng, 2010)

At present, this full-service e-commerce solution, which is generally offered as a self-hosted software, is the most widely adopted implementation model in the market. Nevertheless, the nature of self-hosted implementation imposes some limitations as indicated by Zhonghua & Erfeng (2010). First, due to its implementation difficulties, it requires adequate technical capabilities to deploy and maintain the software, while not all companies have technical staffs capable in this matter. Companies need to install the E-commerce software on their own web server or on rented server. This condition makes the installation more complex and lengthy. Besides that, purchase price of the software is normally quite high, not to mention installation, maintenance and service costs.

These mentioned limitations lead to emergence of a new licensing type of E-commerce platform which leverages the increasing popularity of cloud computing technology and delivered as Software as a Service (SaaS). A SaaS-based E-commerce platform liberates users from costs and technical complexities of installing and maintaining an E-commerce platform since the vendors will take care of all the infrastructure, hardware and software aspects. Users will only need to pay for actual use of the services. SaaS platforms claim to be able to resolve issues of self-hosted e-commerce software mode mentioned above. Users have great flexibility with respect to budget spending as they only need to pay for actual use of the services. Maintenance and upgrade costs are also significantly lower. Users can greatly reduce Total Cost of Ownership (TCO) when implementing their e-commerce solution. Companies also do not need to have extensive technical human resources in place, allowing them to focus more on core business rather than dealing with internal technical issues.

2.3 Features of E-commerce Web Shop Platform

In selecting e-commerce platform to implement, retailers should choose the one that can support basic e-commerce transaction activities as well as their specific business needs. From technological point of view, we could identify some common system components of a pre-packaged e-commerce web shop. The following list of E-commerce features is created by studying from both academic sources and market reports. Depending on the web shop, these features could be either provided natively as the basic packages of the web shop or as separated features by 3rd party service providers.

From academic literatures, Anteneh, Lertwachara, & Thongpapanl (2010) argued that Content Management, Customer (Service) Management and Channel Management are features of an e-commerce platform that have significant impact on online sales. Chu, Leung, Hui, & Cheung (2007) determined core functions of e-commerce platform to conduct e-commerce activities which are grouped into four areas: communication; information presentation and representation; language; storage and retrieval. In addition to the academic literatures, we refer to industry whitepapers from e-commerce platform vendors such as (*IBM WebSphere Commerce*, 2014, *Magento Features List*, 2014, *Trade it ™ Ecommerce Platform Feature List*, 2014). Besides, Sikander & Sarma (2010) from Microsoft prescribed architecture components of e-commerce platform to meet both business and technical requirements.

The following list explains the features that are generally provided in E-commerce platform solutions**:**

- Content & Product Information Management

  Content Management System (CMS) in an E-commerce platform is responsible for storing, versioning and publishing of product content. Using a CMS, brand managers can easily update and publish product content, which can be in the form of text, images, or multimedia. In some cases, CMS is different with Product Information Management (PIM) because CMS mostly deals with managing web content while PIM defines things like product attributes & variations, categorization of product, and even to differential pricing and currencies.

- Website Storefront Management

  Website storefront management deals with aspects related to management of front-end interface of e-commerce website. Through an Administration Panel, administrators can control multiple web shops, customize the design of the overall website as well as individual pages using available templates or their own design.

- Customer Account Management and Customer Service

  Customer account management helps retailers to effectively manage customer through its lifecycle, to increase customer experience and satisfaction. This feature can also work together with existing customer database and Customer Relation Management (CRM) back-office system.

- Marketing, Promotion and Conversion Tool

  In e-commerce era, retailers can't always use traditional marketing and promotional techniques for real-world customers because the customer experience in the digital world is very different. However, this digital era also brings new opportunities to influence buying behaviour and purchase decision of customers. Digital campaigns such as giving a selection of incentives—from highlighted discount offers on specific product pages to integrated e-mail marketing, newsletter, advertising system, search, and portal-based product campaigns, can be implemented. .

- Analytic and Reporting

  Analytic here covers broader scope that traditional web analytic that focuses on website data like web page views and visitors count. Analytic for e-commerce platform, while still provides those traditional analytic functions, also measures performance of marketing effort on traffic, visitors and sales. The analytic results and reports are generally presented in an Administrator Dashboard from which management can be better informed and make better decisions.

- Data Repository & Search

  By default an e-commerce platform should have a robust data management system and repository in place. The transactional data not only needs a reliable database that can store different content and data formats but also a mechanism to query and process the data in multiple ways for rendering and transactional purposes, as well as support high volumes and concurrent processing of transactions.

- Rich Web and Client Presentation

  The presentation part of the e-commerce website can enhance user experience through the use of novel data representation and rendering technologies like Silverlight and Asynchronous JavaScript (AJAX) implemented inside the browser. These technologies collaborate with content management to provide rendering of rich content and media.

- Stable Core Web Framework

  The platform should be built upon a foundation of a robust, extensible, and scalable Web rendering engine that can support both server and client side of modern programming and transaction practices.

- Shopping Cart and Payment (Transaction Processing)

  After customers decide to purchase a specific product, they should be able to make the transaction on the E-commerce website itself. Therefore, E-commerce web shop should provide reliable checkout and payment functions. Shopping cart feature enables customer to eventually purchase the selected items from the website. Customer should also be able to make payment securely using variety of payment gateway options such as credit card, Paypal, or online payment systems like iDeal. The payment should be able to support multiple currencies, tax rates and languages.

- Order Management & Fulfilment System

  A comprehensive order management system may consist of the following functionalities. Retailers can view, edit, create and manage every orders from customers using Administrator Panel. For each order, one or multiple invoices could be created for further process along order fulfilment value chain. Companies can also track the current status of each order along order processing workflow. The system can also make life easier for retailers by automating the order workflow. With respect to inventory management, every sales can be traced and then the real-time stock position or back-order for a certain product can be determined.

- Shipping, logistic/distribution, and warehousing

  To realize the fullest potential of e-commerce, it becomes important to choose the best logistic and fulfilment strategy (Ricker & Kalakota, 1999). Logistic module works hand-in-hand with the order fulfillment module to process the receiving, storing, packaging, and shipping orders to end consumers. Return handling is often provided as part of this module. Generally E-commerce web shop collaborates with logistic service providers to obtain real-time shipping cost and time. Before customer checking out during online transaction, shipping quotes (costs or delivery time estimation) can be already deduced based on a number of factors such as item quantity, item location and customer address. The system can connect to multiple warehouses and multiple Logistic Service Provider (LSP) to automatically check which warehouse with the item is located closest to that customer and which LSP can provide the best service for that specific order. The result is a faster and more efficient picking, packing and delivery process to the doorstep of the customer

- Back-office Integration

  This module is responsible for handling coordination and integration between the (modern) front-end web shop with the existing (legacy) back-office systems. While challenging, it is imperative to support this integration, which generally requires complex network of systems, middleware, databases and applications. However, with recent integration technologies, it becomes possible to simplify this backend integration. Scenarios involving coordination between front-end and back-end systems including synchronization of product data and inventory with data from ERP system or enhancing customer information in the e-commerce database with data from CRM system.

- Social Media Integration

  With the vast proliferation of social media, E-commerce companies become more aware of the needs to use social media, mostly for marketing campaigns. Especially for e-tailers having online store channel in social media like Facebook, this feature could help expand their audience reach, build brand, and obviously grow their sales. It also becomes more convenient to the customers as they can buy goods directly on Facebook online store using their Facebook account, without having to create new account on to the e-tailers website.

- (Multi) Channel Management

  Retailers will be able to manage multiple sales channel through this feature and ensure that their customers have the same experience whether they make purchase in store, web shop, social media or mobile channel. Retailers will also be able to synchronize inventory level and ensure consistent product and customer information across multiple channels

2.4 State-of-the-art of Pre-packaged E-commerce Platform

In this section, a market analysis was conducted to investigate the latest state-of-the-art of e-commerce platform solutions. Through the results of this market analysis, common characteristics of existing e-commerce platforms, functional gaps, and possible future developments were identified. The suitability of any available platform to our case concerning integration capability was particularly assessed.

Based on the types of deployment method explained in the chapter 2.2, the online store platform can be categorized into Self-managed/Self-hosted (on-premise) platforms and Software as a Service (SaaS)/on-demand platforms. For performing this market analysis, we mainly used existing documentations, articles and market reports from industry analysts. For each of the examined platforms, whenever possible, we also conducted a technical "drive test" by trying out the platforms by ourselves. We also investigated websites that already implemented e-commerce solutions under this study and see how they perform.

Some industry research firms have already conducted recent researches about e-commerce platform market. A report by NBS provides a comprehensive comparison of the self-hosted e-commerce solution platforms. In the report, NBS compared 12 different e-commerce solutions based on general Key Performance Indicator (KPI), technical KPI and advanced features KPI. NBS made distinction of the platforms in terms of market segment that they target (From the highest-end market Tier 1 until the lowest-end market Tier 4), the underlying technology (Java or PHP) and software license type (open source or proprietary). (Humeau & Jung, 2013)

According to this report, Magento (owned by eBay) is considered as the one of the most popular e-commerce platforms, with leading position in almost all market segments except the highest end market which is dominated by WebSphere Commerce from IBM. With respect to architecture, it can be differentiated between the enterprise-grade proprietary solutions with their open-source counterparts. Magento, which is supported by a strong open source ecosystem, has quite complete native capabilities while flexible enough to be extended through a wide selection of 3rd party add-ons. Platforms like IBM WebSphere, ATG Oracle or Hybris SAP with their proprietary nature are generally addressing high-end markets by providing more extensive features, better integration and better support. Another factor that might have impact on architecture design is whether the e-commerce platform offered as self-hosted or SaaS (Software as a Service) solution.

A report from Forrester evaluates 10 enterprise-class e-commerce platform providers based on 75 different criteria (Walker, 2012). According to the report, IBM scores the highest in terms of solution architecture (5 out of 5 possible points) and technology architecture (4.5 out of 5). Their solution combines a rich set of eCommerce capabilities with a flexible service-oriented architecture (SOA) and integration capability, resulting in highly flexible and customizable product. Another report from Gartner contains a Magic Quadrant which evaluates e-commerce vendors' Completeness of Vision and Ability to Execute (Sengar, Alvarez, & Fletcher, 2013). In this report, 20 different e-commerce platforms are investigated with IBM, Oracle and Hybris placed in Leaders quadrant. Some key takeaways from this report are: E-commerce is shifting from merely online sales channel to more integrated platform which delivers unified customer experience across multiple channels. Integration needs of e-commerce platform to back-office and external systems are also increasing due to this situation.

2.5 Discussion & Conclusion from Market Analysis

A number of conclusions could be drawn from the literature study and market analysis.As also indicated in the report by Forrester, integration is considered as one of three important requirements of direct-to-consumer online retail business. Aligned to that, retailers have to find a way to integrate the online shop platform into their existing internal systems, which in many cases are legacy monolithic systems, as well as to external systems. In order to work dynamically with multiple partners in the entire e-commerce value chain, retailers also have to deal with disparate information systems of the partners.

The most widely adopted solution by the e-commerce platforms under study for solving this enterprise integration issue is to rely on hard-wired web service based integration. In this approach, each external services is connected to each online shop platform through the so-called “connectors” or “adaptors”. For instance, Magento platform enable users to connect to their ERP system such as SAP by using a “SAP connect” extension. If a connector is not available, some platforms also provide toolkits for users or vendors to develop their own application connectors.

While this approach seems to work just fine, at the end it will produce an inefficient point-to-point integration topology, or commonly referred to as spaghetti architecture. System vendors will have to build custom integration adaptors for each online shop platform. As an illustration, SAP has to provide different connectors for Magento, IBM WebSphere or Demandware. Furthermore, when a retailer wants to switch to another online shop vendor, they will have to start the integration work all over again with totally different type of connectors. This model might work in small company with only a small numbers of applications that need to be tied together. However, when the number of systems to integrate increases, the entire integration schema will be highly complex, which has impact on scalability.

Furthermore, in near future it is expected that cloud computing will gain more popularity as companies and organizations are rapidly migrating their existing local systems to the cloud (on-premise to on-demand). Despite this shift, most of the corporate data remains in on-premise servers due to security and confidentiality constraints. Because of this situation, new integration scenarios emerged that involve both on-premise and cloud based applications (SaaS). It might become cumbersome to integrate systems of different nature like SaaS systems and legacy systems. Connections between SaaS applications are also challenging due to diversity of data models and lack of standardization. (Potočnik & Juric, 2012).

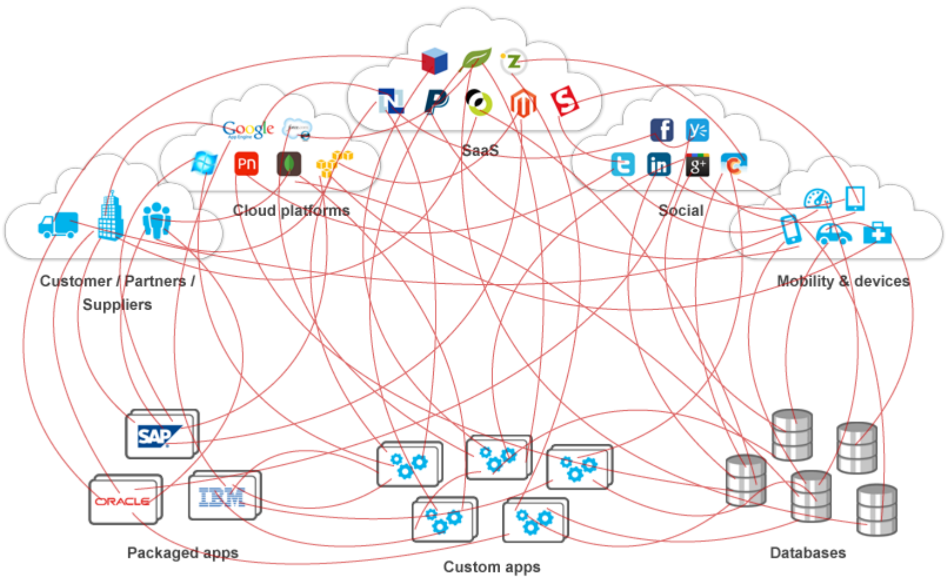


Figure 3 Integration complexity (La Greca, 2014)

The Figure 3 above beautifully illustrates the integration complexity that companies are facing in this digital era. As can be seen in the figure, every systems need to be connected to each other in a point-to-point topology. SAAS applications need to be connected to back-end systems, each cloud platforms need to be linked to each databases, and so on. In order to solve point-to-point integration problem, a middleware solution needs to be implemented. An Enterprise Service Bus (ESB) is a middleware component typically installed in an organization environment applying the Service Oriented Architecture (SOA) design principles to facilitate application integration. Rather than each applications directly connect to each other as shown in the figure above, the applications only need to connect to the ESB through adapters. The ESB will then facilitate communication among applications.

However, conventional middleware technology like ESB is used mainly for internal integration within a company and might not be suitable for cross-organizational integration. When one wants to expose certain parts of an own system and work together with external partners, a special type of middleware needs to be used. Besides, traditional middleware solutions like on-premise ESB is not adequate for supporting integration scenarios that involves cloud services, social media, and mobile channels.

To cope with this issue, a new platform concept has emerged recently: cloud-based service integration platform. This platform offers a more powerful approach to application integration and provide sufficient flexibility to create more complex e-commerce business process scenarios. It provides on-demand integration middleware that enables a wide range of integration and governance scenarios: on-premise to on-premise, SaaS to SaaS, or SaaS to on-premise. This platform also brings advantages in the context of inter-enterprise integration in supply chain. In the next thesis chapter we will elaborate more on this new type of integration platform, both from academic and practice perspective.

3. Integration and Pluggability of Services in E-commerce

The idea behind the pluggable platform architecture presented in this paper is to give users the possibility to integrate services from multiple organizations into the existing environment instead of having all e-commerce functionalities in the basic platform package. This approach will result in an agile architecture that allows more flexible service orchestration and business process compositions with a minimal effort in terms of sourcing and implementation. Thus, pluggability and integration of services can be regarded as related to each other. By ensuring seamless integration, we will be able to add or drop services without having to worry about things like service contract formal definition, technical compatibility or service usage provision. This chapter will explore further about integration aspect in e-commerce supply chain and how it ensures pluggability of services for our platform design.

3.1 Enterprise System Integration in E-commerce

To remain competitive in the current global business, companies should be able to manage and coordinate their activities and complex relationships with their supply chain partners. Prior to achieving successful coordination, it is important to first ensure seamless integration of enterprise applications of the partners. In the context of industrial engineering research and particularly in supply chain management, coordination can be defined as working together among actors in a supply chain to achieve common goal where a decision making process is performed jointly and separate entities influence each other in a more interactive and direct way (Moharana, Murty, Senapati, & Khuntia, 2012). The aim of coordination is to achieve global optimization within a defined supply chain network.

E-commerce is closely associated to Business to Business Integration (B2Bi). B2Bi tasks are challenging because of the diverse and distributed nature of systems in the enterprise network environment, especially for global companies with large number of supply chain partners. B2Bi aims to facilitate communication among the disparate systems and for them to recognize their dependencies with each other, which ultimately will lead to business process automation. The following sub-sections will explain further about B2Bi with the focus on e-commerce industry context.

3.1.1 Enterprise Application Integration (EAI) and Business to Business (B2B) Integration

Integration has been a top priority for companies these days. According to Kurz, Hotop, & Haring (2001), integration scenarios have three fundamental integration problems which have to be solved :

- Data integration, which aims at maintaining consistency of logical data units from a variety of database systems shared by multiple information systems
- Application integration, which supports integration of discrete application logic and functionality to form a coherent aggregate.
- Business process integration, which allows for automation of management, operational, and supporting of multistep business processes. Besides, it also allows for integration of systems and services as well as secure sharing of data across numerous applications

Prior to addressing external integration issues with trading partners, companies must first examine their internal environment to make sure every system works together perfectly (intra-enterprise integration). Typically, legacy systems and back-office systems within company such as Enterprise Resource Planning (ERP) or Customer Relationship Management (CRM) cannot communicate and interoperate with each other. Initially, these information silo are integrated with each other one-by-one, resulting to an inefficient point-to-point integration schema.

Enterprise Application Integration (EAI) technology provides solution to cope with this internal integration challenge. EAI can be defined the process of creating an integrated framework of infrastructure and services for connecting disparate systems, applications and data sources within enterprise environment. By implementing EAI properly, companies could benefit from better information sharing, automated business process, and faster adaptation to market condition. (Samtani, 2002)

EAI is regarded as an initial part of B2B Integration solution, which is the key towards inter-enterprise collaborative e-commerce. B2B integration solution facilitates secured coordination of information, electronic exchange of data and dynamic business process management across business partners and their information systems in order to complete business transactions. When implementing B2B integration solution, companies should be really concerned with aspects like performance, data security, transaction integrity, inter-enterprise business process management, industry standards and internal resistance. With proper implementation, companies could benefit from more streamlined business operation, lower transaction costs, dynamic business relationship and obtaining real-time information. (Samtani, 2002)

EAI and B2B integration are considered different in their nature because the former has been characterized by internally-oriented integration while the latter is external. Yet, they share some common features such as data transformation, messaging, workflow/process management, application-specific adapters, and intelligent routing.

3.1.2 Traditional B2B Integration, XML and Middleware

EDI

In the early phase of B2B integration, one of protocol standards that had been adopted by large sets of corporations was Electronic Data Interchange (EDI). EDI facilitates communication and electronic exchange of routine business transactions among large number of companies over private Value Added Networks (VAN). Such transactions typically includes exchange of highly secure documents like purchase orders, invoices or shipping and payment. EDI formats the documents into compressed, machine readable format which will be then transferred in bulk over the VAN.

EDI is regarded as unsuitable for today’s global business landscape which leverages the use of internet in a more dynamic environment. One major limitation of EDI is its static nature. EDI network should be established with pre-defined set of partners which close its interaction with newcomers. Adding new trading partners is possible but it requires customized mapping to each new partner’s document formats. The initial installation procedure of EDI is expensive, time-consuming and complex, which makes it only implemented in large companies. EDI-based data format is also usually hard to read, expensive, inextensible and batch-oriented. (Samtani, 2002)

XML

Extensible Markup Language (XML) emerged as a solution of the issues associated to EDI. By using XML-based messages, it becomes easier for applications with heterogeneous nature running on disparate type of platforms to exchange the data over the internet and then interpret and act on it. As a universal and flexible language, XML complies with standards of internal legacy systems, back-office systems, application servers and web servers, making their information formatted in a more simple and usable format. (Power, 2005)

XML has substantial advantages over traditional EDI. XML is open, simple and flexible which can be easily read both by machine and human. EDI, in contrast, is very strict and inflexible and can only be read by machine. By using XML, companies also won't have to use specific vendor software as usually required by EDI. XML facilitate semantic agreement between trading partners in the event of XML message exchange. With its powerful meta-language, XML supports development of new markup language for specific industries like cXML (Commerce XML) for e-commerce, as well as domain-specific vocabularies. XML which leverages internet significantly reduces time, cost and complexity of initial set up and operation compared to EDI which uses VAN. As a result, companies have started to migrate their existing EDI-based to XML-based e-business frameworks (Nurmilaakso, 2008).

Despite the drawbacks of traditional EDI efforts, it is currently still the most dominant solution that companies carry out to exchange information electronically. XML was not meant to replace EDI. Rather, EDI and XML have been co-exist for a long time. Typically in large organizations, EDI deals with well-defined, high volume routines while XML is used for new initiatives like B2B participation over the internet.

Middleware

Apart from EDI and XML, another important technology development that can advance the goals of EAI and B2B integration is middleware. Middleware is generally defined as intermediate layer of software that enables interaction, communication and management between applications in a distributed environment. Typically middleware lies in the middle of a client/server system to provide an interface between diverse client and server systems.

Middleware technology can perform the following functions (Turban, Lee, King, McKay, & Marshall, 2007):

- Concealing the diversity and distributed nature of the various operating systems, hardware components, and communication protocols
- Providing a set of common services to carry out general-purpose functions
- Assure simple composition, reusability, interoperability and collaboration between applications through standardized, uniform, high-level interfaces to the application developers and integrators

The role of middleware is critical, especially in highly distributed environment. By integrating heterogeneous systems throughout an organization, middleware technologies can reduce application development time and improve speed to market through standardized development methodologies; reduce maintenance of applications; and reduce latency of data exchange across multiple systems.

Integration Broker
Integration Broker is a special type of middleware for external use which is typically built on top of existing middleware technologies, most often on messaging middleware. Integration broker acts as an end-to-end integration platform which facilitates all types of integration tasks using pre-built application-specific adapters which give bi-directional connectivity to multiple applications. Communication between integration broker and applications takes place mostly in the form of messages. These messages can be stored, searched and retrieved in the integration broker which also serves as a repository.

In a business process workflow, the integration broker will extract data from the source node (which can be an application, a program, or a person according to what is defined in the workflow), then transform it, perform schema conversion and finally routes the data to the target node. Thus, integration brokers can significantly reduce the time and effort required to implement and maintain new business process and application interfaces. They allow enterprises to leverage a single platform and reusable components across multiple applications.

Development of internet
Apart from all the technologies explained above, rapid development of internet technologies in the past several years has led to many new opportunities as well as challenges for integration. XML has been playing an essential role in this Web 2.0 era and has become the most dominant standard facilitating integration among diverse systems. Other interrelated technologies that can prominently support the integration in E-commerce are Web Services and service-oriented architecture (SOA).

It can be imagined in an order-fulfilment case in which for instance, we need to integrate order entry, payment authorization, and shipping software modules which are written in various programming languages and reside on different computer hardware distributed across the Internet. Even when prepackaged E-commerce web shop is implemented, tying those software modules together still requires huge effort, especially if we want to conceal the underlying connections to the end users. Existing technologies also make integration hard because of several reasons such as platform-specific objects, dynamic environment, and security barriers. Due to these reasons, there is a need for universal standards, and this is where XML, Web Services, and SOA come into the picture.

3.1.3 SOA and Web Services
In B2B integration context, support for dynamic integration of heterogeneous applications and platforms across multiple organizations is required. In the event of change in one of the applications, even a slightest change, the change will propagate throughout the entire integration schema. If dynamic integration is not guaranteed, the worst thing that could happen is that the whole integration can be broken. To achieve a real dynamic integration schema, software resources such as applications, programs and objects should be loosely coupled. These resources should reveal their presence to public, describe their actions, and enable communication with applications using them through public interfaces and open standard. In addition, these resources can be tailored for each user to build future applications dynamically in no time.

Service Oriented Architecture

This is where SOA comes into play. Service oriented design is a novel architectural approach in integrating heterogeneous applications and different middleware systems seamlessly across multiple organizations. SOA can be viewed as a set of guidelines, principles and techniques to effectively reorganize diverse software/information systems and the supporting infrastructures which were arranged in silos beforehand into an interconnected portfolio of services which can be accessed using standard interfaces and messaging protocol (Papazoglou & Georgakapoulos, 2003; Papazoglou, 2003).

The term service in SOA is defined as a self-contained module providing standard business functionality which have independency over the context or state of other services (Papazoglou & Heuvel, 2007). By embracing the SOA principle, applications will be decomposed into loosely-coupled services. Each service exposes an interface specifying the operations available and types of messages that can be handled. The services then will be made available to public by exposing them to the internet as Web Services. These web services, which are based on XML, then can be invoked over the internet using XML message and can be installed as local components in a different application.

Web Services

In a service oriented environment, Web Service technology has aroused as a promising way to empower communication, data exchange and integration among disparate and distributed information systems. Web services can be defined as new breed of software objects that can be reused and aggregated over the internet to perform specific functions or to execute business processes in a scalable and flexible manner. Web services are self-contained, self-describing, and modular. Once a web service is published, other applications or other web services can discover and invoke the deployed services. (Fensel & Bussler, 2002)

Web service framework consists of three main components, each with its own specification standard: service discovery, service descriptions and communication protocols (Curbera et al., 2002)

- Universal Description Discovery and Integration (UDDI)
  As the name implies, UDDI allows systematic discoverability mechanism of services over the web by using a centralized services registry. UDDI defines a service registry's structure and operation through two basic specifications: a definition of the information to give about each service, and how to encode it; and a query and update API for the registry that describes how this information can be accessed and updated.

- Web Service Description Language (WSDL)
  WSDL is an XML format developed by IBM and Microsoft, is responsible for describing web services' interface which is used by users as communication point to invoke the web services. WSDL document also tells users what messages must be exchanged to successfully interact with a service. WSDL describes service in both application-level (abstract interface) and specific protocol-dependent details.

- Simple Object Access Protocol (SOAP)
  SOAP is a communication protocol in web service environment used for messaging and Remote Procedure Call (RPC). Because SOAP is based on XML, it offers lightweight, secure, international and platform independent communication mechanism. SOAP works with existing

transport protocols like HTTP, SMTP and MQSeries. In addition, its very simple message structure, SOAP specification also defines how recipients should process SOAP message.

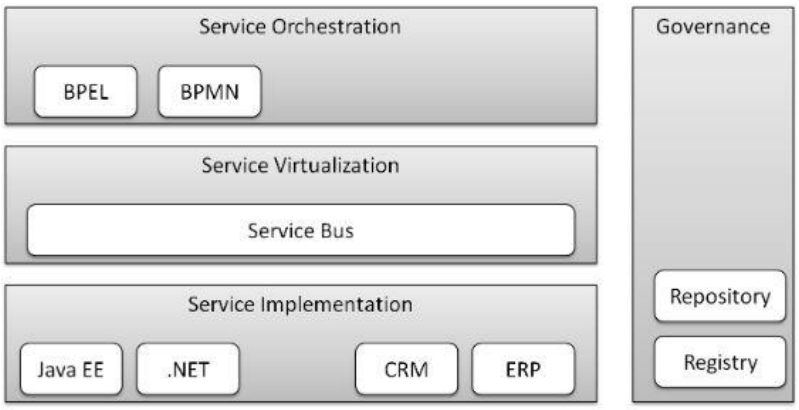


Figure 4 General SOA Stack ("SOA," 2010)

In a SOA-based platform, these are typical components that we can find as revealed in the figure above:

- Enterprise Service Bus (ESB)

ESB acts as a middle integration layer, with reusable integration and communication logic, designed to guarantee interoperability among distributed infrastructures and systems via open & standard-based adapters and interfaces. ESB is a physical implementation backbone for SOA environment with main responsibility to properly control message flow and translations among different services, using any number of possible communication protocols and message formats. Instead of interacting directly with each other, services interact through the ESB which mediates the communication. (González & Ruggia, 2010; Papazoglou & Heuvel, 2007)

- Service registry and repository

Service registry and repository plays a central role in SOA governance (Bertolino & Polini, 2009). SOA Governance defines a comprehensive framework of policies, design rules, procedure and documentation standards which are required for successful cooperation of diverse services, platforms and organizations in SOA environment. Service registry and repository organize information about offered services while facilitating discovery and publication of the services for service requesters. Services, their providers and procedures to invoke the services are described through the Web Services Description Language (WSDL), Universal Description Discovery and Integration (UDDI) standards along with Simple Object Access Protocol (SOAP) messaging.

- Service orchestration

Service orchestration is one of viewpoints of process-based service composition besides service choreography. In service orchestration, process construct is exploited to provide a classical, workflow-style composition which is extended with external message interaction capability. In terms of process coordination across several parties, detailed steps of the process will not be revealed so only those details pertinent to interactions with other processes will be exposed. Conversely, service choreography focuses on message exchange sequences across multiple parties, illustrating a global view of service interactions without showing details of internal processing. (Barros, Dumas, & Hofstede, 2005)

Business Process Modeling (BPM) is one of methods used in service composition. Business Process Modeling Notation (BPMN) is considered as the de-facto standard of graph-oriented language in BPM domain. BPMN provides a graphical notation and specific constructs for BPM through flowchart namely Business Process Diagram (BPD) with an emphasis on control-flow. In addition, there has been increasingly interest to use another standard namely Business Process Execution Language (BPEL) to implement executable business process on top of web services architecture. Due to its open standard and basis on XML, BPEL can operate across disparate environments, making it is suitable for SOA environment, which aims to integrate various different systems by capturing each system as a service performing a specific business function. BPEL facilitates orchestration of both synchronous and asynchronous web services into complete business processes. Thus, there have been some efforts to map BPMN models into BPEL code to achieve unified and standards-based business process development environments. (Ouyang, Dumas, Ter Hofstede, & Van Der Aalst, 2006; Pasley, 2005)

SOA is a suitable framework which establishes a service-based platform as enabler of true and seamless dynamic B2B integration across multiple organizations in supply chain as well as for internal integration purposes (Samtani, 2002). Inter-organizational integration platforms have some distinct requirements compared to systems internally deployed to one organization or only available to a closed business consortium ignore (Hillegersberg, Moonen, & Dalmolen, 2012), especially if the platform aims to act as a one stop shop to source IT services. If the ultimate vision of a service oriented environment is brought to inter-enterprise environment, it means automatic cooperation between enterprises.

With SOA and Web Services, functions within existing enterprise applications such as ERP or CRM, will be encapsulated as services. Service requesters will be able to discover services based on specific requirements like cost or performance, and then invoke the services from anywhere. Developer does not need to know how the services work, only the input that they require, the output they provide, and how to invoke the services for execution. With the use of Web services as an enabling technology, B2Bi related problems and issues will shift from connectivity among different applications in-house and with trading partner applications to the content and structure of the information that is exchanged.

3.2 Latest Technological Developments and Their Implications to Integration in E-commerce

3.2.1 RESTful Web Services & REST API

As the Internet develops and Web applications matures over the past few years, a second generation of Web Services has proliferated and gained increased attention. The web nowadays is increasingly dominated by this new type of Web Services, commonly referred to as Web APIs or RESTful services, which seem to be favored over the traditional ones which rely on SOAP and WSDL because of their relative simplicity and their natural suitability for the Web (Maleshkova, Pedrinaci, & Domingue, 2010). REST web service can be seen as collection of resources which adheres to REST (Representational State Transfer) architectural principle that defines how resources are represented and addressed. REST is based on a set of requirements such as use of a uniform interface, statelessness of the request, and the client-server based communication.

In REST paradigm, applications are connected in a style native to the Web. Therefore, RESTful services rely almost entirely on the use of Universal Resource Identifier (URIs), for both resource identification and interaction, and HTTP for message transmission. REST only utilizes simple HTTP Methods (GET, POST, PUT, and DELETE) to retrieve and manipulate data. This shows how easy it can be to compose or link IT components in a dynamic manner, resulting in relatively simple ways to obtain the original SOA goals of flexibility, reusability, or reduction of complexity. (Su & Chiang, 2012)

RESTful and SOAP Services Comparison

Nevertheless, REST-oriented web service is not necessarily the opposite of its SOAP counterpart. REST is an architectural style while SOAP is a general protocol that can be used as an element of many different architectures. At an abstract level, web services choreography standardization efforts can be categorized into standards that favor a loose coupling of components which adhere to the architectural principle of REST and standards that favor a tight coupling of components using SOAP. Both SOAP and REST have its own advantages and cases where it is more suitable to be used. In fact, some websites provides both interfaces to facilitate web services choreography or other purposes. Amazon.com, for example, figures out that 85% of its developers make use of REST interface while the remaining choose the SOAP interface. (zur Muehlen, Nickerson, & Swenson, 2005).

SOAP-based approach, with its tight coupling of operations, can be tested and debugged before an application is deployed. SOAP relies on formal, well-documented, strictly defined contract in the form of WSDL document. SOAP supports stateful operations, which makes it applicable to asynchronous processes. Nevertheless, SOA requires lots of upfront planning, business modeling, architectural definition, and organization framework in which to operate. In contrast, REST-based system, due to the limited number of operations and the unified address schema, offers high scalability and the lightweight access to its operations. No formal service contract exists in REST as it relies on late binding service contract discovery during run-time. REST only supports totally stateless operations.

In terms of cross-organizational workflows, REST-style workflow utilizes URI to identify each externally accessible process, activity, or operation in a cross-organizational process. Using HTTP GET command to any of these URIs will return a response in either XML document or the more lightweight format JavaScript Object Notation (JSON) object. This response can be cached for later use. While REST relies exclusively on HTTP/HTTPS protocol, SOAP has more options as it works with almost any transport protocols to send requests. SOAP wraps every requests and responses (SOAP Headers, XML Tag) with XML. Caching is also not supported by SOAP. This makes the lighter REST approach is preferred particularly by mobile devices and applications. In addition, the HTTP POST command is typically used by client to send message containing HTML form parameters or payload to the RESTful services. The POST command being responsible for manipulating the state of the process instance until it is completed. This combination of the GET and POST commands is the key to this subsequent interaction between client and process instance.

On the other hand, in the classical SOAP-style process integration, the endpoints of the communication are described in WSDL while SOAP is used as messaging standard. In that sense, every operation is represented by its own communication endpoint instead of a message type. For instance, consider an example scenario of a SOAP-style purchasing process containing a purchase order object. Because every possible operations that can be executed on that object is represented through its own endpoint, client can initiate creation of the object by invoking the ‘‘CreatePurchaseOrder’’ operation. Afterwards, the purchase order instance can be exploited through explicit operations such as ‘‘UpdatePurchaseOrder’’ and ‘‘CancelPurchaseOrder’’.

SOA Governance and API Management
Both SOA Governance and API Management is regarded as essential components to enable pluggability of services to our platform design. SOA Governance and API Management basically share the same underlying architectural design principle, which is service oriented design. Both aim to govern and manage the service lifecycle including design, implementation, publication, operation, maintenance and retirement of services and APIs (Malinverno, Plummer, & Van Huizen, 2013).

SOA governance technologies, however, have been around for several years and almost reached maturity. SOA governance is mostly about managing services within an organization, even though in some cases the services can also be exposed to business partners. SOA governance covers a wide range of functions including but not limited to policy enforcement, security, service contract, compliance, Service Level Agreement (SLA), lifecycle management, service registry and repository (Schepers, Iacob, & Van Eck, 2008).

On the other hand, although API Management comprises of similar building blocks of SOA Governance, it involves some distinct capabilities (Maler & Hammond, 2013). It can be said that the fundamental difference of API and SOA lies in its orientation of service consumption. SOA tend to be geared towards internal consumption while API, due to its openness, can be used both internally within company and by external developers. As a consequence, some additional components, such as enterprise gateway, security, developer portal, and service billing need to be in place. Through APIs, companies can consume external services without having to first form formal partnership agreements with service providers. Companies only need to follow the authentication and authorization procedures and then subscribe to the API that they want to consume. By being authenticated and authorized, it becomes easier for the API provider in tracking API usage and accordingly, charge companies based on their API usage.

By examining the current internet landscape, it can be seen that more and more REST APIs have been made available to public. E-commerce domain is without exception. Organizations within e-commerce fields have started to release their APIs to facilitate direct access to the functionality provided by their applications, which in turn will leverage third-party efforts to add value to existing services (Foping & Walsh, 2013). Furthermore, due to the rise of mobile apps in recent years, the trend of mobile commerce also proliferate which comes along with demand for new lightweight model to consume enterprise backend functionality. REST-style approach can open doors to more capabilities of enterprise backend. By exposing the multiple, heterogeneous backend sources as RESTful Services, enterprise can unlock their backend data and functionalities to a wide range of mobile solutions & web applications. Aside from that, through the use of lightweight, familiar standards like XML and JSON, enterprise can better engage with (external) mobile and web developers as well as drive more innovations from them. By enabling integration with mobile applications, true multi-channel e-commerce experience can be achieved.

3.2.2 Cloud computing & SaaS

The vast development of cloud computing technology in recent years has substantial impact to Information Technology landscape as more and more organizations begin to adopt this technology. Cloud computing can be defined as both the applications delivered as services over the Internet and the hardware and systems software in the data centers that delivers those services. (Armbrust et al., 2010).

One of the important characteristics possessed by cloud computing is elasticity, which is the ability to dynamically scale up or scale down computing resources whenever required to match with system workload within a very short time frame (typically within minutes). Through elasticity, users of cloud computing could avoid risk of over-provisioning (underutilization) and under-provisioning (saturation). Other notable characteristics including on-demand self-service, resource pooling, and multi-tenancy (multiple customers can use the same computing infrastructure and network, which result in increase of utilization rate). (Jula, Sundararajan, & Othman, 2014)

Three different cloud computing service delivery models could be distinguished (Jula et al., 2014; Rimal, Jukan, Katsaros, & Goeleven, 2010) as presented in Figure 5:

- Infrastructure as a Service (IaaS):

  In this case, vendor only provides the basic infrastructure such as networking, storage, servers and virtualization because the consumer has developed the required applications on their own. IaaS mainly benefits enterprise users as they do not have to spend much money on initial investment and ongoing maintenance of IT infrastructures. Users can also enjoy the latest technology in the market without having to buy it themselves.

- Platform as a Service (PaaS)

  Vendor provides a cloud based platform in this delivery model. The platform basically consists of all the systems and environments, ranging from hardware infrastructures, operating systems, as well as application development tools and user interfaces, which allow the consumer to manage applications throughout their life cycle of developing, testing, deploying and hosting.

- Software as a Service (SaaS)

  SaaS is a distribution model for software or application that is hosted on a vendor's infrastructure and made available to consumers as a service. As depicted in Figure 5, both packaged software and SaaS cover the entire system from networking level until application level. However, in SaaS, the vendor manages the entire system instead of the user as in the traditional packaged software. Due to multi-tenancy nature, SaaS shares common infrastructures and resources to simultaneously support multiple consumers, but the consumers are unaware of the provider's infrastructure. Consumers also have limited permission to access the service and limited authority to configure some settings. The provision is performed through a thin client like a web browser or a program interface for sending data and receiving results.

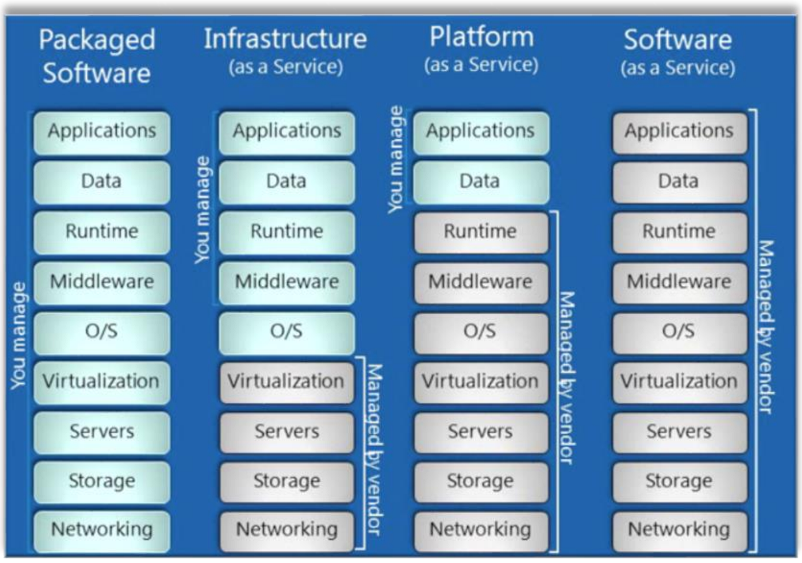


Figure 5 Cloud Service Delivery Model (Czernicki, 2011)

In the context of application integration, on one hand, integration involving cloud-based applications possesses new integration challenges due to its different nature with legacy systems and on-premise applications. Besides the traditional integration scenario between on-premise systems, new integration scenarios such as cloud to cloud integration and cloud to on-premise integration proliferate as a result of high adoption of cloud computing technology (Kleeberg, Zirpins, & Kirchner, 2014).

As also argued by Rimal et al. (2010), the major considerations in SaaS are the integration requirements with other applications. Data from SaaS needs to be consolidated and synchronized automatically with on-premise applications. In application integration level, even though generally SaaS relies on simple and standardized interfaces and technical standard to exchange data and logic, some Application Programming Interface (APIs) and web services are specific to individual SaaS or cloud platforms and usually APIs do not provide all needed functionalities, which makes integration more complicated (Cusumano, 2010; Potočnik & Juric, 2012).

On the other hand, cloud computing also bring novel ways to solve the above mentioned integration challenge. Traditional middleware and integration platform, which are typically deployed as on-premise systems, could leverage cloud computing technology to obtain benefits of cloud-based platform. The next section will elaborate more on this cloud-based integration platform.

3.2.3 Cloud-based integration platform

Cloud-based integration platform is a novel type of system integration platform delivered as a service over cloud (Kleeberg et al., 2014). This platform operates as an on-demand middleware that facilitates development, execution and governance of integration flows connecting any combination of on-premise, cloud, mobile and social media systems in many-to-many fashion, both within the same organization and across multiple organizations.

Integration flows here can be defined as custom-developed software and metadata employing the integration logic required to make possible the interoperability of multiple applications. Some actions that are needed to be carried out in integration flow include message transformation, routing, protocol conversions, service virtualization, orchestrations, security federation, usage tracking, administration, monitoring and management. Users of the platform access the cloud-based development environment and accordingly, they can develop integration flows by themselves. The platform also constantly secure, manage and monitor integration on data, application, services and business process level.

Cloud-based integration platform vendors typically also provide governance platform services which include registry/repository, artefacts life cycle management, policy management and enforcement, as well as the extraction of the associated data. Strong tie between integration and governance capabilities is the best practice of SOA design which is still considered valid in the era of cloud computing. Through its governance platform services, cloud-based integration platform enables both design time and runtime governance of the integration process models, artefacts, and other relevant integration components. Since this platform resides on cloud, it also inherits the benefits of cloud technology as compared to traditional on-premise middleware and integration platform. Users won't need to build and manage hardware and software infrastructures as they are provided by cloud-based integration platform vendors, resulting in lower integration costs. Users also benefit from cloud computing technology such as flexibility, agility, high availability and scalability.

In e-commerce B2B integration context, it is a widely adopted and proven concept to run integration flows on a third-party integration platform provided as a service. Suppliers, business partners and customers then can better collaborate. New e-commerce trends like social and mobile commerce could also come into the picture. Being in the cloud, this platform has close proximity with other cloud-based services and SaaS applications, make integration becomes less daunting task.

Figure 6 below shows relation of cloud-based integration platform with on-premise and SaaS applications. As can be seen in the figure, the heart of the platform is the lightweight, cloud-based ESB which facilitates behind the scene communication between the platform and on-premise as well as SaaS applications through the dotted line. Lightweight ESB is more advantageous than classical enterprise-grade ESB which is complex, expensive and not easily deployed to a cloud environment. Yet, the main drawback of lightweight ESB is its lack of support for complex, long-running business process.

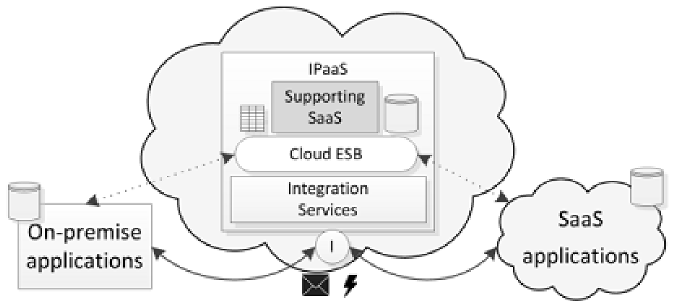


Figure 6 Cloud-based integration platform ecosystem (Potočnik & Juric, 2012)

Companies that already have an on-premise integration platform (such as an ESB) in place, can implement cloud-based integration platform along with its on-premise counterparts to get the best of both worlds, forming the so-called hybrid cloud integration platform (Ried, 2014). In this case, internal integration within the enterprise is managed by the on-premise platform while external or B2B integration is taken care of by the cloud-based platform.

Considering the benefits of integration platform explained above, we consider cloud-based integration platform concept as the most suitable platform architecture approach to support pluggability of services. The next chapter will provide our market analysis of this type of platform. Through the market analysis, we could derive best of breed architecture approach and components, common features and functionality gaps. The results of this will be a basis our architecture design in chapter 4.

3.3 State-of-the-art of Cloud-Based Service Integration Platforms

The cloud based integration platform is a relatively new concept and commonly referred to as Integration Platform as a Service (iPaaS), a term coined by Gartner (Pezzini, 2011). IPaaS is anticipated as the next generation of integration as a service, which has been widely adopted for e-commerce B2B and cloud services integration domains. These domains are the most-proven use cases for iPaaS so it is advised to initially implement iPaaS in e-commerce B2B scenarios (Pezzini & Lheureux, 2011).

A recent study by Gartner evaluated and compared Enterprise iPaaS providers (Pezzini et al., 2014) . In the Magic Quadrant produced in the report as shown in Figure 7, three platforms are placed into Leader quadrant, which are Dell Boomi Atom Sphere, Informatica, and Mulesoft Anypoint Platform. Mulesoft Anypoint Platform is interesting because of its strong open-source ecosystem and its core technologies which combine on-premise integrations using Mulesoft ESB, cloud integrations using CloudHub and API Management capabilities through Anypoint API Manager. Dell Boomi and Informatica, with their proprietary nature, offer rich Cloud Service Integration, B2B, on-premises application integration, API and Mobile Application Integration (MAI) functionality if compared with other players.

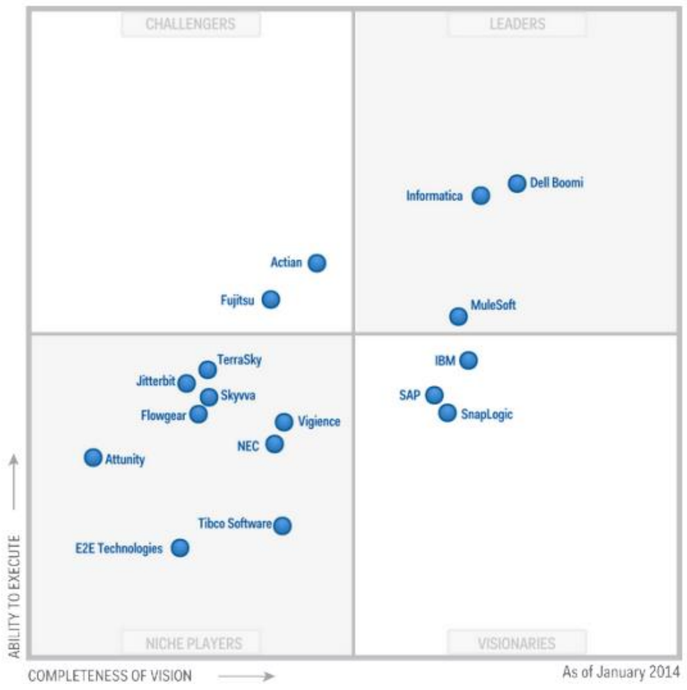


Figure 7 Gartner Magic Quadrant for Enterprise iPaaS (Pezzini et al., 2014)

Another research by Forrester assesses 14 vendors providing hybrid integration solutions, which in their description, comprise of four integration scenarios: on-premise integration, cloud-based integration, iPaaS and API Management (Ried, 2014). The author argues that hybrid integration capabilities are not simply exposed by new cloud based integration or iPaaS products. Rather, there is also an increasing capability set of established and traditional integration tools. Dell Boomi, Informatica and Mulesoft again become the leading platforms for all four scenarios. The only platform that share the leader title is IBM.

3.4 Discussions & Conclusions

Several conclusions could be drawn upon from the results of the market analysis. Based on our investigation, a number of common features are typically provided in the basic package of iPaaS solutions. To create and manage the integration flow, iPaaS vendors usually provide an Integrated Development Environment (IDE) which simplify development of integration flows through a visual drag and drop user interface. This feature lets both technical and non-technical users to perform standard integration tasks. Pre-built connectors or adapters are a must, especially for common enterprise systems like SAP or Paypal and to popular cloud services like Amazon or Salesforce, to let users to immediately construct integration scenarios and reduce development time. If a connector for a certain application is not already available in their repository, some vendors provide a connector software development kit (SDK) for users to develop custom connectors.

Besides IDE and connectors, other essential automated components that typically have to be provided to connect applications across different environments are a data mapper, data transformer, and endpoint components. Vendors also tend to incorporate API Management capabilities into their platform, in addition to SOA Governance, to deliver a complete solution to deal with both on-premise and SaaS. Some vendors also offer pre-packaged integration flows that are commonly performed such a recruiting processes or order fulfillment.

Table 1 and Table 2 provides list of meta-services architecture component that have been derived based on the findings of this chapter. We make distinction between “Service Framework” and “Process Framework”; the former deals with services throughout their lifecycle while the latter is responsible for managing integration process flows. These two frameworks will be the main component of our platform which will be elaborated more in chapter 4.

Table 1 “Service Framework” meta-services

“Service Framework” meta-services	Explanation
Developer Portal	In the developer portal, companies should provide relevant and comprehensive aspects of their APIs such as API documentation, policy, terms and agreement, testing environment (sandbox or real), or API versioning
Enterprise Gateway	Manage the interaction between the API and external API consumers
Policy Enforcement & Management	Manage both design time and runtime policies of services. Design time policies are concerned with things like design guidelines or security mechanism while run time policies are concerned with operational environment and requirements that have to be met by the service at runtime.

Security	The different between security in SOA Governance and API Management is that in SOA Governance, the organization administer internal and known users while API Management handles external & unknown users. API Security manages additional aspects like authorization and authentication, API Key management, as well as Identity and Credential Management.
Service Analytic & Reporting	Obtain insightful traffic analytics and reports of API activities with respect to developers account, application, or services as well as observe overall API usage and trends
Service Level Agreement Management	Service Level Agreement (SLA) Management concerns with guaranteeing the service level as stated in SLA contract, service evaluation as well as fees for consuming the service and fines if terms in the contract are violated.
Service Lifecycle Management	Lifecycle Management is concerned with managing the design, development and delivery of individual services in a SOA. The tasks include change management procedure, service registration and even deciding on service granularity.
Service Metering & Billing	Monitor and measure service usage as the basis for billing calculation for the service consumers. Service performance is also monitored regularly
Service Registry & Repository	Service Registry can be viewed as a catalogue of services that manages the publication of services and define taxonomies of the published services using which consumers can find suitable services to their needs. While the Service Registry only contains service references, Service Repository is the actual holder of documentation, policies and metadata about versioning of the service.

Table 2 “Process Framework” services

“Process Framework” services	Explanation
Development and Lifecycle Management Platform Services	Manages service integration process flows throughout their lifecycle from modeling, development, configuration, testing until deployment.
Integration Platform Services	Consists of aspects that ensure seamless integration flow both at design time (service orchestration) and runtime (process execution). These aspects include but not limited to : Message/Data transformation and routing, Integration Development Environment (IDE), Adapters, Flow Management, Protocol conversion, Service Virtualization, and Security federation
Monitoring, Management, and Administration Platform Services	Takes care of deployment and administration of integration flows, monitor their execution and manage their behavior. Covers several aspects such as Technical and Business Activity Monitoring, Logging and tracking. and Error resolution

Apart from the common features of iPaaS platform described above, common missing parts were also identified. The existing iPaaS solutions typically lack of intuitive and easy to use drag-and-drop integration flow creation facilities without too much technical settings, for instance like those offered in the consumer market (e.g. Elastic.io or IFTTT). Widely adopted industry standards for business process modeling and execution such as BPMN & BPEL are not always supported; the graphical tools tend to use vendor-specific modeling instead of using vendor-neutral notations. Complete integration extensions for the entire e-commerce value chain are also not available, which results in incomplete support for every possible e-commerce specific business processes, especially those related to logistics and order fulfilment.

To sum up, according to the results of our market study, we could infer that current iPaaS solutions could become a suitable solution for the integration issues of online shop platforms. Nonetheless, this generic integration platform might not be able to address all possible business process scenarios and requirements for a certain industry, or in our context, e-commerce industry. In the next chapter, we propose a more specific reference architecture of an integration platform specific for e-commerce

4. Reference Architecture Design and Development

The previous chapters have motivated that the predominant way to add more functionalities into modern enterprise architectures is to outsource the functionalities through third party service providers. Chapter 2 and 3 have covered architecture components of e-commerce web store platform and integration platform both from academic and practice point of view. The results are used as inputs for designing the reference architecture in this chapter.

Section 4.1 sheds a light on the definitions of reference architecture. Section 4.2 then elaborates the architecture design principle, framework and modeling language that we choose. Section 4.3 presents detailed design for each of three business actors that we consider as important in our architecture design. Section 4.4 is the culmination of all previous sections in which the complete reference architecture is constructed by combining the three separated architecture models in section 4.3. Finally, chapter 4.5 will provide discussions and conclusions to this chapter.

4.1. Definition of Reference Architecture

IT solutions that solve a specific business problem are often derived from reference models of the business in question, which tend to be both generic (to meet the requirements of all kind of business cases) and detailed (to serve as a blueprint for system implementation) (Fettke & Loos, 2006). Companies that adopt standard model based solutions can also use the reference models to analyze the fit and gaps between their current processes and the process model. Creating a reference architecture is useful as a mean to give a common convention of system structures, elements and relation among them in the context of specific domain, which in our case is for E-commerce field.

Because of numerous definitions of what a Reference Architecture in IT domain actually is, it is better to start by providing a clear definition of it. Architecture is defined by IEEE (2000) as "the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principle guiding its design and evolution". Another work by Bass, Clements, & Kazman (2003) defines a reference architecture as a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them. A reference model divides system functionality as pieces and define data flow between them while a reference architecture provides mapping from the functionality onto a system decomposition.

Similarly, Enterprise Architecture (EA) can be defined as a comprehensive set of principles, methods, and models used to design and implement the entire elements of an enterprise, covering from the organizational structure, business processes, information systems, and IT infrastructure (Jonkers et al., 2006; M. Lankhorst, 2005). EA provides an integrated and holistic view of the enterprise and translates enterprise business strategies into technology solutions to ensure optimal IT investments which makes it suitable to deal with business and IT alignment problems.

4.2. Enterprise Architecture Design Principle and Modeling Language

Service-oriented is regarded as a suitable design principle to create flexible E-commerce solutions which support seamless & dynamic integration of heterogeneous systems, both within the organization and across multiple organizations. Systems within SOA environment are encapsulated as services which are then published through open, standardized interfaces. This approach enables simple way to communicate with the services. The loosely coupled nature of SOA design paradigm makes each services independent of each other. Service contracts can then be updated without having to always interact with the service consumer.

When a retailer adopts a specific e-commerce platform solution, it is nearly impossible for the platform to be the industry-leading expert in all the E-commerce functionalities that have been identified in chapter 2.3 Features of E-commerce Web Shop Platform If the functionalities are tightly coupled to the platform, the options for the retailer are severely limited. On the contrary, if each of the functionalities are treated as a separate service and if the core platform enables true pluggability of services, retailer could easily customize the platform to meet their specific needs by adding or removing services into the platform. As an example, retailers can add a Product Information Management function from one of the leading PIM service providers in the market while removing social media integration function from the platform because the retailer does not have any social media accounts.

In fact, there have been a number of researches attempting to use service-oriented design and web services as the core technologies to transform a static supply chain scenario into a dynamic, “loosely coupled” business network that can be easily adapted and configured. One of the most notable researches is the research by Hillegersberg, Boeke, & Heuvel (2004) which developed a supply-chain scenario by using Microsoft Biztalk as the Web Services orchestration tool and BPEL Web Services orchestration. Based on the experience from the research, it was concluded that SOAP and WSDL standards were able to support a true cross-platform distributed architecture. Given that these web services standards are well supported across platforms, straightforward B2B integration can be achieved through standard and low-cost Internet technology.

Moreover, a concept that is closely related to architecture design principle is architecture framework. Enterprise Architecture Framework (EAF) outlines relation and interaction of all of the software development process within the enterprise in order to help organization identify and understand weaknesses and inconsistencies that might hamper organization in achieving its objectives (Urbaczewski & Mrdalj, 2006). The framework generally proposes design principle, method and set of tools to assist design of architecture building blocks. While Enterprise Architecture describes a broad structure of business entities and processes as well as IS and IT to support them, a unified framework will help enterprise in narrowing the gap between IT and business domains in organizations. By using an appropriate framework, planning, designing and implementation costs could be reduced through more efficient process of architecture design, evaluation and building.

Zachman Framework by IBM is one of the most well-known frameworks. Presented in 1987 by John Zachman, it can be considered as the very first enterprise architecture framework. Constructed as two-dimensional matrix, the framework consists of five rows and six columns which represent perspectives and aspects. The five perspectives are contextual (scope), conceptual (enterprise/business), logical (information system), physical (technology), and as built (deployment) while the six aspects are what (data), how (function), where (network), who (people), when (time) and why (motivation). An intersection of a row and a column shows an answer for each question to each perspective. (Zachman, 1987)

The Open Group Architecture Framework (TOGAF), which is developed by the Open Group consortium, is one of the most widely adopted architecture framework standards in industry. The core of TOGAF is Architecture Development Method (ADM) which describes an approach to design, develop, implement and evaluate an enterprise architecture. Four architecture domains are included: Business architecture (organization and its key process), Data (the structure of both enterprise's logical and physical data assets), Applications (the individual application systems as well as their relationship and interaction to the business processes), and Technical (the sets of hardware, software and communication infrastructure which support the deployment of the previous three architecture domains). (TOGAF, 2009)

Many of the enterprise architecture frameworks have differences in terms of their approach. Some frameworks proposed specific methodologies and aspects to embrace while others provide guidelines. With respect to level of detail, most of the frameworks provide general and abstract terms, which makes the validity or the ability to work accurately within that framework in doubt. Tang, Han, & Chen (2004) evaluates and compares six different frameworks by examining their goals, inputs and outputs. According to the results, they conclude that TOGAF is able to fulfil almost all their criteria, which makes it regarded the most complete framework.

In addition to a framework, a modeling language is needed in providing a unique visualization of all enterprise architecture domains, relationships and dependencies, thus complementing the used framework in developing the desired enterprise architecture. Selecting a suitable modeling language is imperative to ensure successful development of enterprise architecture. There are several modeling languages that have achieved wide recognition in industry. Unified Modeling Language (UML) is a general purpose modeling language to create a visual design of software systems which has become the de-facto standard in the field of software engineering. UML defines several diagram types which cover the entire aspects of an enterprise architecture, from business process, interaction, environment, system structure to physical development.

Archimate is an open and independent enterprise architecture modeling language which has seen a widespread adoption. Similar with TOGAF, Archimate is developed by The Open Group and is supported by numerous tool vendors and consulting firms. While TOGAF recognizes relevant architecture building blocks and prescribes the modeling process in detail, it is not a complete language as a notation for the building blocks is not provided. Here is the part where Archimate and TOGAF complement each other to make up a complete, powerful and integrated approach for delivering enterprise architecture. They both have a strong common ground as they both share view on the use of viewpoints and repository of architectural artifacts and models. ArchiMate defines a well worked-out language, including a (graphical) notation to provide a concrete visualization for the architectures and views proposed in TOGAF.

As shown in Figure 8, Archimate has a layered pattern which consists of Business, Application and Technology layer and three columns namely Information, Behaviour and Structure distinguishing the relationships and activities performed by the actors of the organization. The Business Layer encompasses products and services offered to external customers, which are realized in the organization through business processes performed by business actors. The business layer is supported by the Application Layer through software applications as realizations of application services. Infrastructure services that are required to support applications are included in the Technology Layer. (The Open Group, 2013)

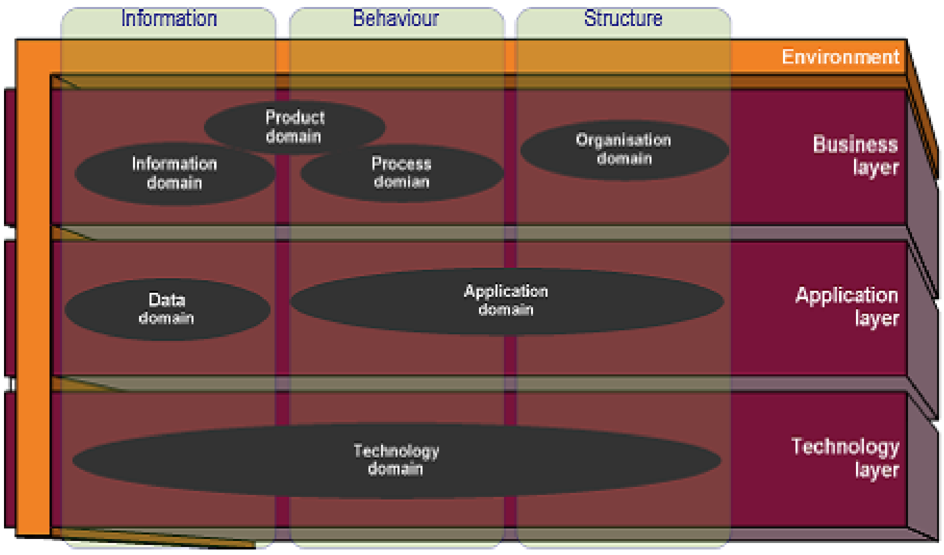


Figure 8 Archimate layers

4.3 Reference Architecture Development

In designing a service platform, the core activities to perform are to design the services and the general platform components supporting the services. Consequently, general requirements that will be incorporated in the services platform should be carefully defined and organized. Besides the requirements, the platform should have configurable and extensible characteristics in order to cope with huge numbers and variations of applications as well as future developments. (Costa, Pires, & Sinderen, 2005)

The pluggable service platform proposed in this research will be designed using Archimate modeling language. Archimate meta-model is presented in Figure 9. For each layer, there are several artifacts to be connected with each other by embracing certain design principles and rules.

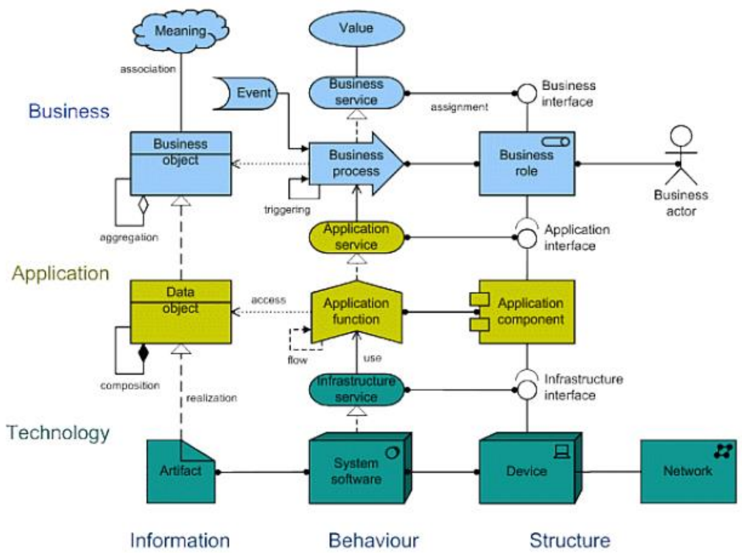


Figure 9 Archimate metamodel (The Open Group, 2013)

In this research, the architecture will be organized around Business Actors (an organizational entity that is capable of performing behavior) with each business actor having separate part in the complete architecture to give an overview of how they collaborate through services in order to achieve common goals. Based on literature review of business process of E-tailers, we could identify three different Business Actors which contribute to the value creation of the platform: The (Online) Retailer who is using the platform to execute the online retail process described above; the Business Service Provider who is using the platform as a means to expose their services to customers; and the Platform Provider itself who acts as an intermediate between service providers and consumers.

The results from Chapter 2 and 3 make up most of the contents in this chapter and will be used to decide upon what to be included in the architecture. Chapter 2 which covers state-of-the-art study of general E-commerce platform functionalities will be used for Online Retailer role, as well as for Collaborative Data Management component design in Platform Provider role. Chapter 3 which explores the best-practice of (cloud-based) Integration Platform will serve as the basis for Collaborative Service and Process Framework component in Platform Provider role, which contains the desired key functionalities and integration support requirements for the integration platform. For the Service Provider role, we identify several type of services considered as relevant in the field of e-commerce.

4.3.1 Online Retailer

The online retailer is the seller of goods, partially or exclusively over the online channel. Figure 10 shows our proposed architecture for this particular actor. The presented architecture for the retailer actor can be considered as the current state of the art and does not introduce any new concepts in itself. In that sense it is a starting point for the use of a pluggable service platform. The architecture should allow for a gradual transition from current, rather monolithic landscape to a cloud service based architecture. It should be possible to add services to that landscape and successively shift new and existing functionality from internal systems to the cloud.

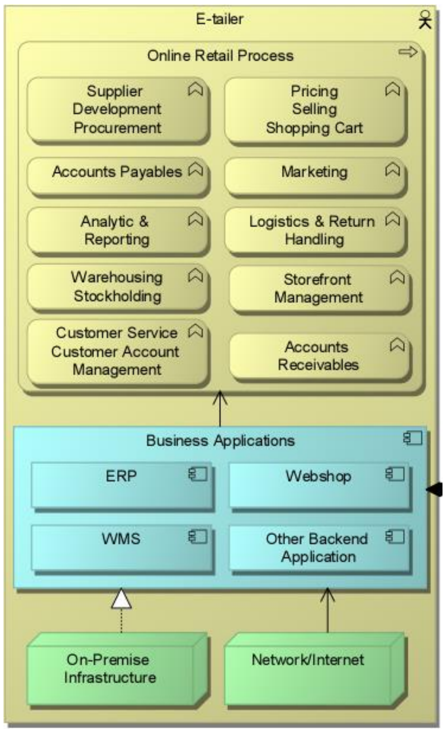


Figure 10 Proposed Architecture of Online Retailer

Business Layer

In the business layer, to specify the requirements for a comprehensive e-tailing platform in a top-down approach, a high level description of the value chain and internal business functions is required (Fettke & Loos, 2006). Although there is no description of a process specific to online retail business model can be found in the academic literature, various contributions provide references on how such a process should look like. For instance, an unpublished work by Aulkemeier, Schramm, Iacob, & van Hillegersberg (n.d.) makes comparison among four online retail process models and combine them into one model to get an overall set of primary business activities in e-tailing. With this model at hand, gaps between literature and practice can be identified when it comes to online retail business process activities. Business functions required in order to fulfill the online retail process steps are then identified by referring to the e-commerce functionalities that have been identified in chapter 2.3. These functions are then grouped into distinct functions to be incorporated in the architecture design.

According to Specification of Archimate 2.1, a business process is defined as "a unit of internal behavior or a collection of causal (sequence or dependency) related units of internal behavior, with the goal of producing a predefined collection of products and services". A business process can be constructed from sub processes or activities which are executed in a certain sequence and can be triggered by other business processes or one or more business events. Every activity in a business process is part of a business function. A business function is interpreted as "an area that the organization wants to pay attention to in order to support its business goals". We can view a business function as a collection of internal behavior according to certain criteria such as same department, required skills, or shared resources. Both business process and business function describe internal behavior of performed by a business role. (The Open Group, 2013)

Application Layer

Depending on the business model and size of the retailer, different business Application Components will be implemented on the application layer. In Archimate specification, an Application Component is regarded as "a modular, deployable, and replaceable part of a software system that encapsulates its behavior and data and exposes these through a set of interfaces". Since it is a self-contained unit of functionality, Application Component can be independently deployable with its re-usable and replaceable characteristics.

A retailer coming from an offline channel business with a number of brick and mortar stores will possibly already have an Enterprise Resource Planning (ERP) installed to manage its operations. When introducing an online channel, the retailer will add a web shop to the landscape that allows customers to browse and order goods online. The order fulfillment and other back office activities will be carried out by the ERP system. Thus, the e-commerce platform in this case consists of a lightweight web shop and the ERP system. In contrast, a pure online retailer might implement a more comprehensive e-commerce platform. Those platforms not only provide a web shop but also a rich set of back office functionality. Depending on the complexity and size of the business, an ERP component might not be present at all because it functionalities have been handled by the web shop.

Beside the web shop and the ERP system, more specialized components might be present on the application layer. If the warehousing operations are not outsources to a fulfillment center, the retailer will probably have a warehouse management system (WMS) in place to manage movement and storage of materials in warehouse. Besides, other essential systems might also be present, for instance Customer Relationship Manager (CRM), Supply Chain Management (SCM) or advanced analytics system. The details for each mentioned systems above are out of the scope of this research.

Infrastructure Layer

From technical point of view, all these applications in the upper layers can be either operated as on-premise or SaaS solution provided in form of web applications by a service provider. Accordingly, either on-premise or network infrastructure have to be deployed on the technical layer of the architecture.

If retailer decides to deploy its webshop as on-premise applications, they should prepare some traditional IT infrastructure components including:

- Application server, to support both Front Office operations (activities focusing on customers such as sales, marketing and customer service) and Back Office operations (activities used by employees such as administration, finance, inventory or order management)
- Database Management System (DBMS), which is assigned to the application server. DBMS in minimal provides CRUD (Create, Read, Write and Update) data service
- Web Server
- Data Storage and Backup
- Security system (firewall)
- Middleware, to connect all nodes together through communication paths
- Communication infrastructure (LAN, network/internet, wireless communication)
- Additional nodes (Disaster Recovery, Proxy Server, Load Balancing)

Basically it could be said that retailer will need to have all required infrastructures in place as well as internal human resources to deploy webshop as on-premise application. Despite the high initial investment and maintenance of on-premise deployment, not to mention its typical lengthy implementation, retailer will have complete control of the entire system and data. The specific deployment infrastructures are out of scope of this study so all the nodes are encapsulated as "On-Premise Infrastructure" instead in our architecture.

On the other hand, if retailer choose to adopt SaaS solutions instead, they do not have to provide most of the infrastructures by themselves since they are provided by SaaS vendor that they choose. Retailer will still need to invest in fast and reliable network infrastructures as all activities are completed over the internet or mobile device. Because the infrastructure is managed by third party entities, the downside is that retailers have limited control on the system and data. Yet, retailers have benefits from significantly lower investment costs and faster implementation.

4.3.2 Service Provider

The service provider can either issue pure ICT services or be a supply chain partner that provides business services (B2B e-commerce). Both service types of service providers have to be integrated on information system level for seamless process execution. As depicted in Figure 11, only application layer is presented. The reason behind this model is that we regard service that service providers offer are in the level of application, not business. We would like to demonstrate how all those services can support business layer of online retailer. Infrastructure layer is also not shown because we are not interested in seeing the infrastructure in detail and how the infrastructure support the provided services.

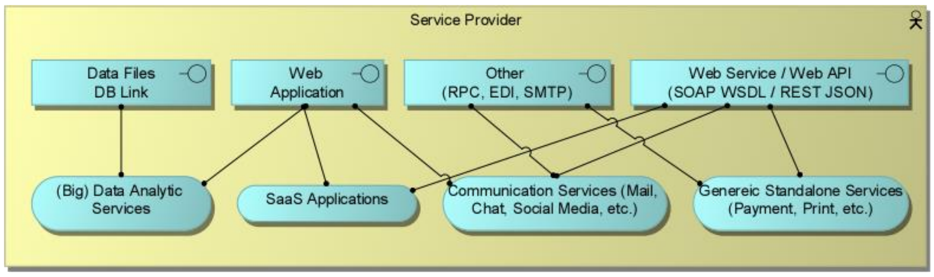

Service Provider
Data Files DB Link
Web Application
Other (RPC, EDI, SMTP)
Web Service / Web API (SOAP WSDL / REST JSON)
(Big) Data Analytic Services
SaaS Applications
Communication Services (Mail, Chat, Social Media, etc.)
Genereic Standalone Services (Payment, Print, etc.)

Figure 11 Proposed Architecture of Service Provider

Application Layer

An application service is defined as an externally visible unit of functionality, provided by one or more components, which is made available to the environment through well-defined interfaces. The definition implies the relation between application component, application function, application interface and application service. An application interface is specified as a point of access which is assigned to an application service to expose the service to its environment, which could be a user or another application component. The same application service may be exposed through multiple interfaces. An application function is defined as a behavior element that groups automated behavior that can be performed by an application component. An application function describes the internal behavior of an application component. If this behavior is exposed externally, this is done through one or more services. An application function abstracts from the way it is implemented sonly the necessary behavior is specified. (The Open Group, 2013)

The actual application services offered by the service providers can be either additional components that internal systems/business functions of retailer cannot cover or functions that are outsources for strategic reasons. The actual services as well as its granularity are too diverse to provide a comprehensive list. Effectually, it should be possible to integrate any kind of service through the architecture. However, the most important aspect here is to obtain a comprehensive picture of potential service interfaces a platform needs to support.

Based on reviews of various architectures in the domain, we identified five different types of IT services to be provided in an e-commerce scenario. The type of service determines through what kind of interface it can be integrated. Through application interface, a kind of contract that other application components have to fulfil to access functionality of a specific application component is determined. The contract itself contains some parameters, protocols used, pre- and post-conditions, and data formats.

The five types of services are: (Big) Data Analytic Service, SaaS Applications, Communication Services (Chat, email, social media), Generic Standalone Services (payment, print), and (Master/Transaction)

Data Services. In addition, those types of services will be connected to four types of application interfaces: Data Files (DB Link), Web Application, Web Service (SOAP WSDL)/Web API (REST JSON) and Other (RPC, EDI, SMTP).

- Data Analytic Service

  On the backend, data analytical services will be integrated through the Data Files interface type that allows exchange large data amounts. As message based integration produces a big overhead, it is not suitable for such services which will rather extract and load data through batch files or database links. Data analytics services often require a large amount of data such as orders and customers that can only be provided through a file transfer or a direct link to the containing database, as a data exchange through messaging would be too resource intensive. The output of the data analytics can be made accessible through web applications.

- SaaS Applications

  Web interfaces (Web applications and web services) are generally used as user interface and used for SaaS, social media services and analytical services and reporting in form of dashboards.

- Communication Services

  Another interface type is based on more specialized protocols that can be considered as more ancient way to integrate services. Despite their higher complexity and technical dependencies, those protocols are still widely used to integrate legacy systems or communication services such as mail and chat but will not be implemented by modern SaaS applications. Communication services such as e-mail, chat or social media will communicate through web clients, APIs or dedicated protocols such as SMTP.

- Generic Standalone Services

  Message based integration can be realized through modern web services or web APIs that communicate over HTTP and can be consumed with state of the art integration tools and techniques. This kind of interface is suitable for standalone services such as payment services, address verifications, customer or credit enquiries but also to access or populate resources of SaaS applications and social media services in a programmable manner.

- Master/Transaction Data Services

  Special attention has to be paid for Master/Transaction Data Services (which is not shown in the diagram). Data services that are used to store or provide that is used across applications (e.g. customer or product data) in a centralized manner (master data management) will be interfaced through web services, web APIs or EDI in case of legacy systems. This service is realized in the Platform Provider section, so we will learn about it in detail in the next section.

4.3.3 Platform Provider

The pluggable service platform acts as an intermediary between the retailer and the service provider. The goal of the platform is twofold: it should allow retailers to source ICT services from third party providers instead of having all the needed functionalities served as native capabilities in web-shop; and the platform should enable retailer to collaborate with supply chain partners.

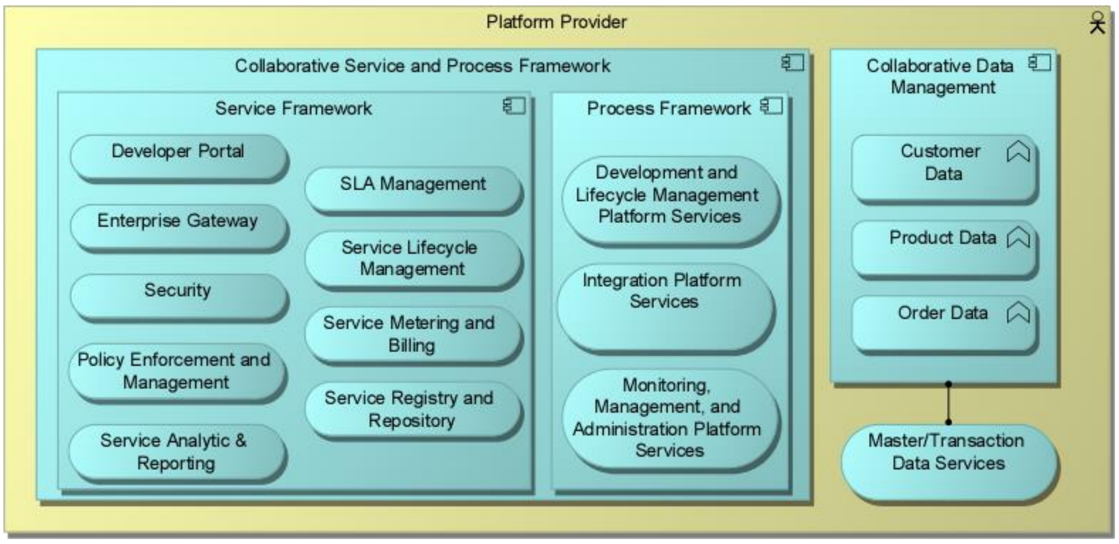

Platform Provider
Collaborative Service and Process Framework
Service Framework
Developer Portal
SLA Management
Enterprise Gateway
Service Lifecycle Management
Security
Service Metering and Billing
Policy Enforcement and Management
Service Registry and Repository
Service Analytic & Reporting
Process Framework
Development and Lifecycle Management Platform Services
Integration Platform Services
Monitoring, Management, and Administration Platform Services
Collaborative Data Management
Customer Data
Product Data
Order Data
Master/Transaction Data Services

Figure 12 Proposed Architecture for Platform Provider

In Figure 12, the only layer displayed is the application layer. Business Layer is not displayed because we are only concerned with application services which will be orchestrated to carry out business process. Infrastructure layer is not depicted either because how exactly these application services are deployed and supported by the infrastructure is out of the scope of this reference architecture. Even though we are not interested in the detailed view of infrastructure layer of the platform provider, typically this layer consists of middleware node representing a hub and spoke solution to allow communication between all other infrastructure nodes. The Enterprise Service Bus is an example of such a middleware implementation. This enables communication between applications without requiring from each of these applications to know the communication details of peer applications, which in the end avoiding point-to-point system topology.

Application Layer

Two key components in this layer are the Collaborative Service and Process Framework and Collaborative Data Management. Derived from the findings in Chapter 3, the Collaborative Service and Process Framework component encompasses:

- A Service Framework, which borrows its functionality from SOA governance and API management, providing the meta-services to maintain services throughout their lifecycle;
- A Process Framework to develop, execute, analyse and monitor service-based process flows

The Collaborative Data Management component, as the name suggest, provides a central repository of both master and transaction data which can be accessed by the E-tailer, its business partners and service providers to be used in distributed service-based business processes. This component exposes its functionalities as the Master/ Transaction Data Service.

Master data denote a company's essential basic data which remain unchanged over a specific period of time such as customer, supplier, or product data. Master data serve as the basis for handling business processes, which in today's world could span across multiple organizations involving heterogeneous information systems whose master data are often neither current nor consistent between systems (Loser, Legner, & Gizanis, 2004). Master data should be consistent no matter who access the data, from where and how the data is accessed by what devices. Inconsistent master data cause process errors and thus higher costs.

On the other hand, transaction data are data that describes events that occur within the business, for instance data related to order delivery, sales or other events. Thus, master data can be viewed as nouns (business objects) while transaction data as verbs (action performed to the business objects). Transaction data are often categorized into financial (involves orders, invoices, payments), work (plans and work records) and logistic data (deliveries, travel records, etc). Master/ Transaction Data Service ensures consistent, unified view of data across all partners.

To facilitate agile and seamless business process integration, a canonical data model has to be established. Canonical data model serves as a single standardized representation of data format that is shared, understandable and agreed on by all applications (Wick, Rohanimanesh, Schultz, & Mccallum, 2008). As one of enterprise application integration patterns, it requires every applications and services involved in the business environment to work with this common format. Implementation of the canonical data model eliminates point-to-point translation among all individual data formats for each application. Rather, the translations will occur between each data format to the canonical data format. This approach delivers full control and visibility of data as well as streamline data synchronization over the entire business process across multiple organizations.

The usefulness of canonical data model becomes apparent when a canonical message schema is also incorporated into the picture. Using this standard schema, which can be an industry-wide or organization-specific standard schema, the sender and receiver of a message will have shared understanding of data type, range of values and semantic meaning, resulting in easier integration of data from different sources (Bernstein & Haas, 2008). The data model in this study has been designed by referring to a standard E-commerce model in ("E-commerce Data Model," 2009).

Schema mapping and schema matching are some of the core technologies of canonical schema. Schema mapping operation translates schema from the data source to conform to the target schema. To illustrate, in the data source, customer name is recorded as First_name and Last_name. However, in the target schema the customer name is recorded as Customer_name, which is the full name of the customer, combining his first name and last name in one field. Thus, this translation should be defined manually in the schema mapping tool. Schema matching is a more sophisticated technique than schema mapping. Schema matching uses machine learning techniques to find reasonable match for elements from one schema to another using any available information such as similarities of name, structure or data-type. For instance, using this algorithm, Employee_Name data field in Schema 1 can be considered similar with EmpName in Schema 2. However, human input is still needed from user to validate the automatically generated matches.

In our design, the Collaborative Data Management component is realized by Cloud-based database services, which is commonly referred to as Database-as-a-Service (DbaaS). One of the main benefits of cloud platforms lies in the centralized storage of resources. Having files or media content stored in a shared repository is beneficial for reuse throughout systems and collaboration among different parties. While the present cloud integration platforms discussed in Chapter 3 shift existing integration middleware features to the cloud, they do not embrace this key benefit of having shared resources among systems and collaboration partners. The reason for this shortcoming lies in their generic nature and the platforms which fail to identify the resources. The domain specific platform architecture will therefore define the resources that are crucial throughout all transactions in a retailing process. Besides, DBaaS inherits pretty much the same benefits of outsourcing IT Infrastructure to the cloud such as flexibility, scalability, and lower investments.

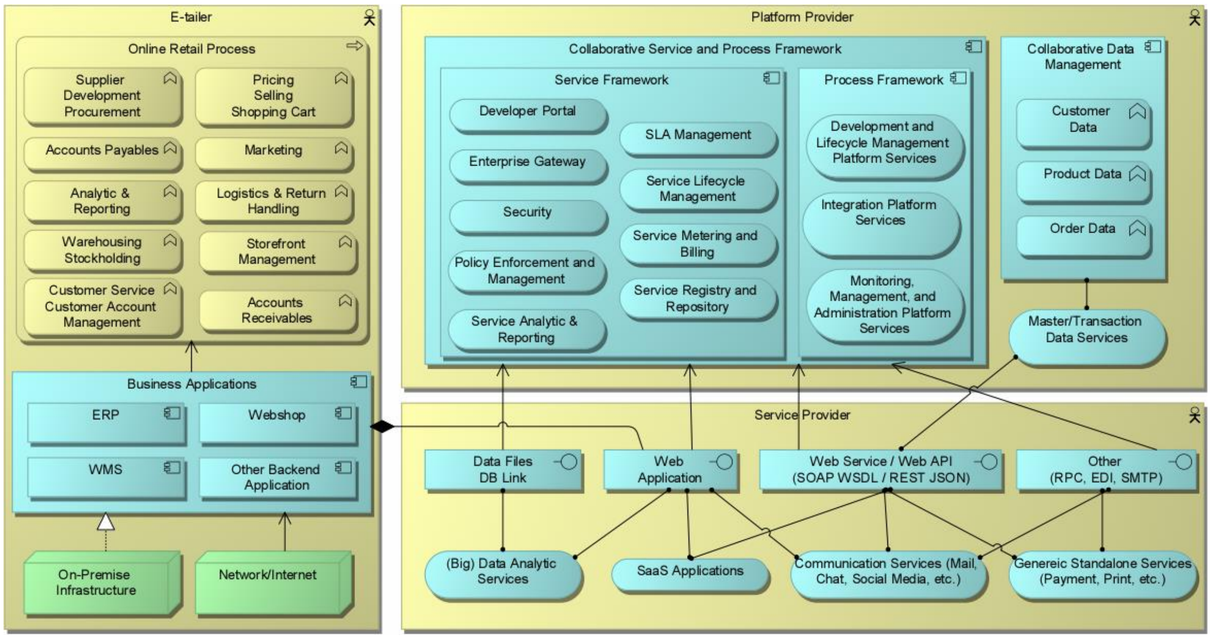
Nevertheless, cloud database is perceived to be having security issues such as lack of privacy, which can reduce trust of users. Data security in the cloud could be preserved by meeting security requirements such as Data Confidentiality, Integrity, and Availability (Almulla & Yeun, 2010). Data confidentiality means that the data in the cloud can't be accessed by unauthorized party. One way to achieve this is to implement proper encryption mechanism. Data integrity means that the stored data is accurate and consistent throughout its lifecycle. Last, data availability ensures the stored data is available when requested by authorized users.

Identity and Access Management (IAM) becomes more important in the cloud to guarantee that the right entities accessing the right resources for appropriate reasons at the right time. IAM is considered as a suitable method to provide Authentication, Authorization and Auditing for users accessing data stored in the cloud. Authentication is the action of verifying the identity of users or systems. After successful authentication, Authorization determines privileges to be given to authenticated users. Last, Auditing step process of reviewing whether the authentication and authorization comply with predefined security policies and standards or not. (Almulla & Yeun, 2010)

In order to facilitate the secure delegation of access rights and permissions to business partners, a widely adopted solution in the industry is to use OAuth protocol. OAuth is an open authorization protocol standard which provides generic framework which can be used by resource owner to allow third party entities to access the owner's resources without having to expose the owner's personal credential like username or password to the entities.

4.4 The Complete Reference Architecture

In this section, all architecture models from previous sections will be combined into a unified model to get a clear overview of the whole platform. *Figure 13* illustrates the complete picture the proposed reference architecture of pluggable service platform for e-commerce.


E-tailer
Online Retail Process
Supplier Development Procurement
Pricing Selling Shopping Cart
Accounts Payables
Marketing
Analytic & Reporting
Logistics & Return Handling
Warehousing Stockholding
Storefront Management
Customer Service Customer Account Management
Accounts Receivables
Business Applications
ERP
Webshop
WMS
Other Backend Application
On-Premise Infrastructure
Network/Internet
Platform Provider
Collaborative Service and Process Framework
Service Framework
Developer Portal
SLA Management
Enterprise Gateway
Service Lifecycle Management
Security
Service Metering and Billing
Policy Enforcement and Management
Service Registry and Repository
Service Analytic & Reporting
Process Framework
Development and Lifecycle Management Platform Services
Integration Platform Services
Monitoring, Management, and Administration Platform Services
Collaborative Data Management
Customer Data
Product Data
Order Data
Master/Transaction Data Services
Service Provider
Data Files DB Link
Web Application
Web Service / Web API (SOAP WSDL / REST JSON)
Other (RPC, EDI, SMTP)
(Big) Data Analytic Services
SaaS Applications
Communication Services (Mail, Chat, Social Media, etc.)
Genereic Standalone Services (Payment, Print, etc.)
Figure 13 The complete Reference Architecture of Pluggable Service Platform for E-commerce

As portrayed in the overall architecture, the retailer part does not mean that the retailer has all the functionalities in place. Rather, because of agile nature of the proposed e-commerce platform, the platform will only have small set of core capabilities which then can be extended by plugging in additional services into the platform. For instance, Warehousing and Stockholding function could be obtained from third party service provider. As a result, retailer will end up with lean platform and will have the flexibility to tailor their platform according to business needs. The link between Business Applications component on Retailer side and Web Application interface on Service Provider side denotes the fact that Retailer incorporates additional services to the Business Applications.

Between the Service Provider and the Platform Provider, two types of links could be observed. The links between Application Interfaces on Service Provider side and Collaborative Service & Process Framework component on Platform Provider side illustrates that the platform manages services according to their interfaces. Moreover, application interface is used to access functionalities of component exposed by application services. Web Service / Web API interface type is used to connect Master/ Transaction Data Service because it is a common practice to access functionalities of cloud database using REST API, which produces responses as XML or JSON data. Together with authentication protocol like OAuth, theoretically they form a seamless and secure approach to access functionalities offered by Collaborative Data Management component.

5. A Service Platform Based Return Registration Process

Based on the propositions in the previous section, the implementation of the service-oriented platform design is demonstrated in this section by realization of a prototype for a specific case in the e-commerce value chain. Return registration process is selected as the use case to implement. There are a number of arguments behind this decision.

Returns handling process is vital in an online channel retail scenario, since the efficient handling of return shipments increases customer satisfaction and can lead to major cost savings. While varying and depending on the country and type of good, and often kept undisclosed by the retailers, return rates of 35% and higher seem to be realistic in some sectors (Denery, n.d.). With that high volume of return shipments, the effective handling of reverse logistics, refurbishment and other related tasks become almost as crucial as the order fulfilment and should be regarded as a regular process that has to be handled systematically as a primary business activity.

Return management function is not provided in the basic package of almost all e-commerce platforms in this study. Some platforms provide this functionality through 3rd party module. As the reference model in retail usually do not cover return handling processes, it can be assumed that retail information systems are not naturally designed for handling return shipments efficiently. This condition is supported by research from van Hillegersberg, Zuidwijk, van Nunen, & van Eijk (2001) which stated that even though the management of return flows had become a strategic plan in a growing number of organizations at that time, there was only little attention paid to this aspect by the Information System field. Return handling function was not commonly incorporated in the supply chain management system or ERP system, which tend to focus more on the forward flows. It was also stated that due to the uncertainties introduced by return flows, more coordination and collaboration of business partners in the supply chain is required. This statement makes it even more appealing to select return handling as the use-case for the prototype.

This section is constructed around the concept of Model Driven Architecture (MDA). To begin with, the concept of MDA and its relevance in the context of software development i explained. Subsequent sections are then structured according to three different models of MDA. Following the guidance described by Streekmann, Steffens, Claus, & Garbe (2006), the business process covering a part of the overall return handling process is constructed first as the Computation Independent Model (CIM). Then, the specific architecture for return registration solution is displayed as the Platform Independent Model (PIM). Last, a prototype is created using one of available integration platforms in the market as the Platform Specific Model (PSM).

5.1 Model Driven Architecture and Service Integration

Model Driven Architecture is a framework for design and development of software systems. MDA prescribes a set of guidelines by capturing different aspects of a (distributed) application into symbolic arte-facts known as models. Models are manipulated through-out the design process resulting ultimately in one or more realizations of the application (Almeida, Eck, & Iacob, 2006). These models are expressed using well-defined notations at different amount of system detail, highlighting specific viewpoints or aspects (Mellor, Scott, Uhl, & Weise, 2002). A series of transformations can be established between the models to move back and forth between various levels of abstraction. By using

MDA, it is expected that the time and cost of the software development process could be significantly reduced.

According to the specification of from Object Management Group, MDA comprises three main layers (Miller & Mukerji, 2003): The computation-independent model (CIM) is the top layer which represents the highest level of abstraction of model of the system describing its domain. This stakeholder-oriented model focuses on environment and requirements of the system. Layer in the middle is called platform-independent model (PIM). The focus of this visual-based conceptual model is in the operation of the system while details for a particular platform are not revealed. The model here is typically defined in diagrams using standards like unified modelling language (UML). Finally, the layer in the bottom is platform specific model (PSM). Within this model, specifications from PIM are combined with details of how the system uses a specific platform. A PSM combines the specifications in the PIM with additional details that specify how that system uses a particular type of platform. Through this transformation from PIM to PSM, implementation method of MDA, which is regarded as model-drivel development (MDD), occurs here.

Application of service oriented principles alone to Enterprise Architecture is considered not adequate to address the integration and interoperability issues in spite of its significant contribution (Jardim-Goncalves, Grilo, & Steiger-Garcao, 2006). One of the reasons behind this argument is that there is no unique standard of Service Oriented Architecture (SOA) implementations which hampers interoperability between these implementations.

In order to solve this situation, Jardim-Goncalves et al. (2006) proposed an integration of SOA and Model Driven Architecture (MDA) in their research. MDA proposes an open approach to write specifications and develop applications while making distinction between application & business functionality with the platform technology. MDA also brings model driven approach to bridge the gap between business level and information system level (Touzi, Benaben, Pingaud, & Lorré, 2009) by using the method, language and tool required to model the desired artifacts practically (Khoshnevis, Shams Aliee, & Jamshidi, 2009). SOA complements to this through establishment of a software architectural concept with the main idea to encapsulate disparate software systems as independent services and provide standardized way to access the services. They claimed that by implementing SOA and MDA architectures together, company's competitiveness could be increased due to adoption of a standard-based extended environment which will result in improved interoperability.

5.2 Computation Independent Model & Business Process Diagram

The first task to accomplish in the MDA development process is to create the CIM. The CIM is highly significant because it serves as the foundation for subsequent development stages and changes in it will propagate to the rest of the development stages (PIM and PSM). Therefore, it is strongly suggested to develop CIM in collaboration with domain experts. CIM comprises the domain specific processes, which in this context is the return registration process.

In the scenario, an end customer should be able to register a return request online. After logging in through a web page in the webshop, the user can select an order and obtain information on the items contained in the order. He then can choose which order-lines/items and what amount he wants to return for each item. Optionally, he can specify the reason for the return as well. The return request is then transferred to the retailer who will decide whether or not the request will be approved. If not approved, the customer will be immediately informed by email. If approved, the return handling is then planned by registering the expected return to database and ascertaining the appropriate return center. The shipping service provider is contacted to generate a RMA (Return Merchandise Authorization) label for print and nearest address of drop-point based on the customer address. This information, together with the label is forwarded via email to the end customer who can prepare the good for shipment and deliver it to the destination prescribed in the RMA label.

Figure 14 below portrays the business process of return registration that has been explained above. Business Process Model and Notation (BPMN) is implemented to construct the process model with Bizagi Business Process Modeling (BPM) tool.

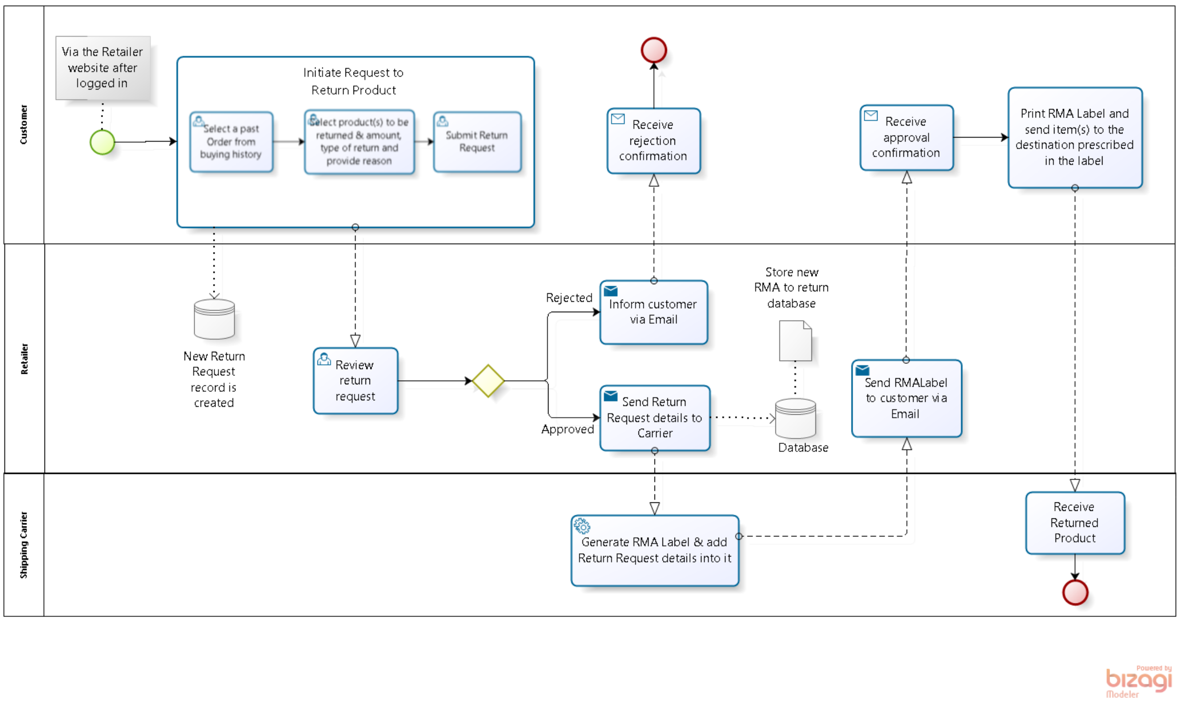

Customer
Retailer
Shipping Carrier
Via the Retailer website after logged in
Initiate Request to Return Product
Select a past Order from buying history
Select product(s) to be returned & amount, type of return and provide reason
Submit Return Request
Receive rejection confirmation
Receive approval confirmation
Print RMA Label and send item(s) to the destination prescribed in the label
New Return Request record is created
Review return request
Rejected
Approved
Inform customer via Email
Store new RMA to return database
Send Return Request details to Carrier
Database
Send RMALabel to customer via Email
Generate RMA Label & add Return Request details into it
Receive Returned Product
bizagi Modeler

Figure 14 Process Model of Return registration Authorization

5.3 Platform Independent Model & Return Handling Architecture

In this section, an Enterprise Architecture revolving around return handling business process will be constructed. The BPMN model in the previous section serves as the foundation of business process in the architecture. Similar with the Reference Architecture in chapter 4, the return architecture is also constructed using Archimate language.

Figure 15 shows the overall architecture of the solution. The pluggable platform operates the collaborative service flows (the blue part) that make use of the various services to provide the business functions with the required information. As seen in the model, each and every service relies on the collaborative resources required to fulfill the service. The return SaaS solution requires information about the orders made by the customer in the past and to register the return. The same applies for the LSP and the ESP that require information on the customer. Having these resources in the platform allows adding and exchanging services to the overall process in a more flexible way.

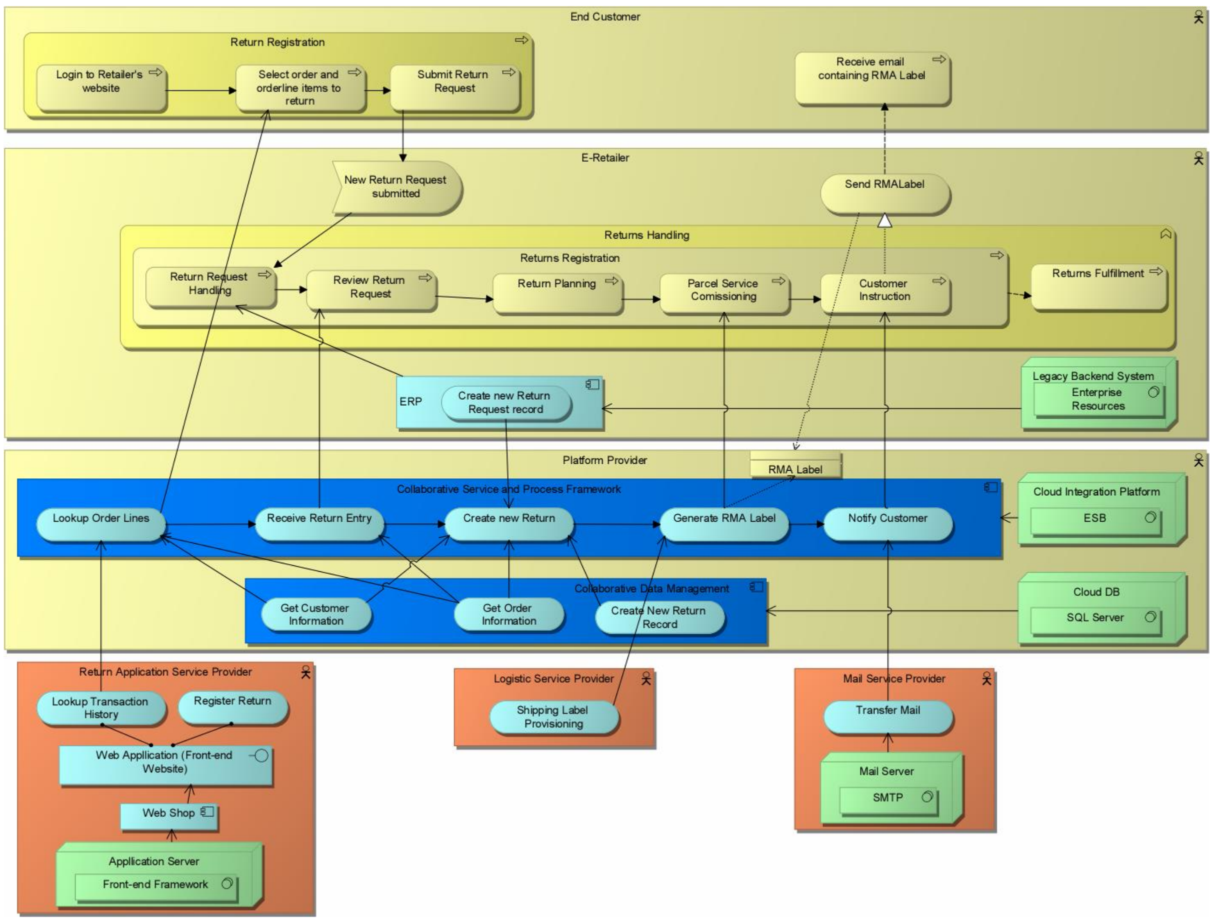


Figure 15 Architecture of Return Process

On the business layer of the architecture (the yellowish part), we focus on the return registration process. The end customer is provided with two services to register returns and receive a return label from the parcel service to send in the goods. We assume that this functionality is not contained in the frontend and backend systems of many retailers.

At the infrastructure layer we are assuming a common setup of backend and frontend systems, as well as a mail server and an application server for custom applications providing data, application and e-mail messaging services to the application layer. For our scenario the integration platform is added to provide service composition and a connector framework as described in the previous section. We also add the cloud services to the infrastructure layers that will be consumed by the connectors although they are external to the organization and do not require any resources.

5.4 Platform Specific Model & Mulesoft Integration Flow

In this final stage, the return handling architecture presented in the previous section is translated into the Platform Specific Model (PSM) by realizing the architecture as an integration flow built on top of a specific integration platform that has been selected. The services in our example are implemented using different technologies, are distributed among different environments, and utilize different protocols to communicate with other systems. Figure 16 gives an overview about the diversity of technologies, platforms and communication protocols used by the services.

Starting from the top left component, a web application portal serves as the main interface for customers to register their return request. This web application communicates through REST services with the platform the fetch order and customer information. Then, following counter-clockwise direction, the Collaborative Master Data services realize their connection to the service platform through REST API Endpoints that we have developed. A REST-based service is again invoked to handle shipping related tasks, or more specifically in this case is to generate the RMA (Return Merchandise Authorization) label. A SaaS Business Process Management (BPM) tool permits design and execution of complex business process workflow over the cloud through API, giving high flexibility and complementing the previously mentioned REST services. Finally, an SMTP (Simple Mail Transport Protocol) is implemented to allow sending and receiving of email. It can be noticed that none of the service here use SOAP as the underlying technology.

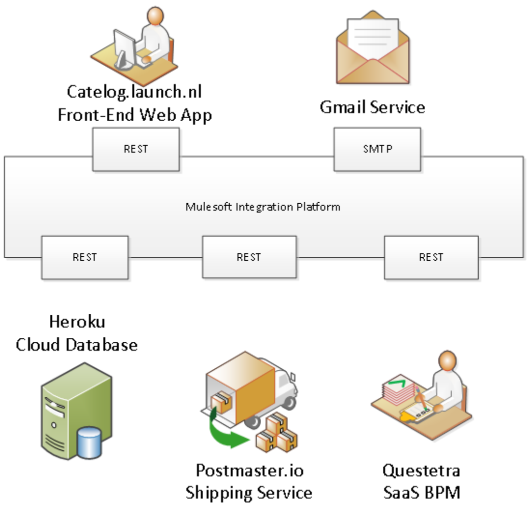


Figure 16 Services and Communication Protocols

In this case, theoretically REST-style is more preferred in terms of pluggability because, thanks to the use of URIs and hyperlinks to identify services, it becomes possible to discover Web resources without an approach based on compulsory registration to a (centralized) repository (Pautasso & Leymann, 2008). This implies limitless possibilities as service consumers can invoke literally any REST services, not bound to only services available in the service repository as in SOA environment. The implementation of REST web services choreography then becomes imperative to achieve true automated integration and coordination across institutional boundaries.

Mulesoft Studio is selected as the integration platform to be used in this research to support the Collaborative Service Flow. On a side note, recently the name of the platform has been changed to Anypoint Studio, but we will still refer it as Mulesoft Studio in this research because this name is already well-known and also to make it easier to search information on the internet. Mulesoft Studio is placed by Gartner & Forrester as one of the leading integration platforms, sharing this title with tools from big names such as Dell Boomi, Informatica, and IBM WebSphere.

Mulesoft Studio has its own strengths because of its open-source nature, strong community, strong ecosystem & support from third party service providers. It also complies with most of the integration platform application components/functionalities as indicated in Collaborative Service & Process Framework part of the proposed architecture chapter 3. Mulesoft Studio is an eclipse-based Integrated Development Environment (IDE) which allows users to create integration process flows using graphical design environment, resulting in faster and more powerful way to perform integration tasks. Developers who are already familiar with Java environment and Eclipse IDE will be able to get used to this mode-driven integration platform quickly.

5.4.1 Catelog Front-end Webshop

A front-end portal website shown in Figure 17 serves as the main interface through which customers can initiate a request to return a product that they ordered. A fictional E-commerce company namely Catelog is used in this case. First, customer needs to enter his order ID. If the submitted number is correct, which means there is a record in the database of that ID, the page in Figure 18 will be displayed.

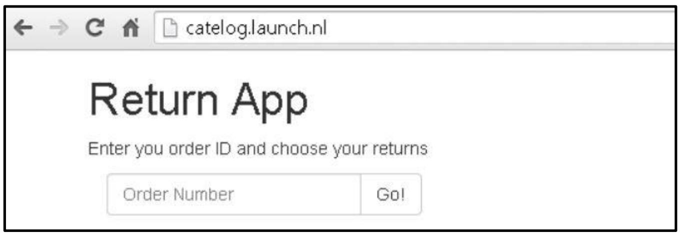
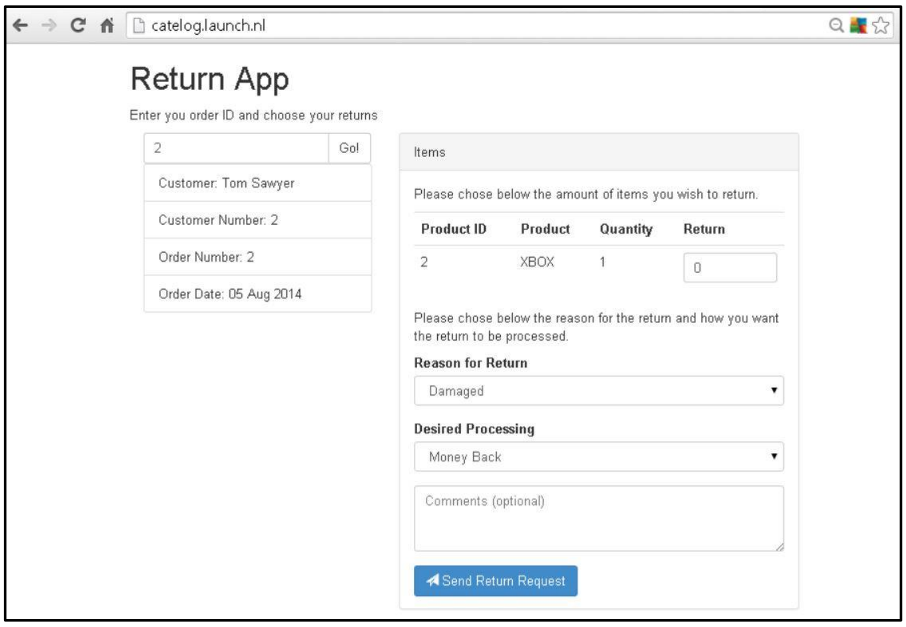


Figure 17 Catelog E-commerce Return Portal

On the left part this page, customer details are presented. On the right part, details of each item associated to that specific order number are displayed such as product id, product name, and quantity of the items that have been ordered. Customer then can fill in how many of the ordered product that he wants to return in the "Return" text box. Underneath the item section, customer can select one of several reasons of why he wants to return the product from the first drop-down menu (in the *Figure 18*, "Damaged" is selected), type of return request ("money back" is selected in the figure), and optionally provide comment. It should be noted that the options in the dropdown menus are populated from database data, not hard-coded.



Customer: Tom Sawyer
Customer Number: 2
Order Number: 2
Order Date: 05 Aug 2014

Items

Please chose below the amount of items you wish to return.

Product ID	Product	Quantity	Return
2	XBOX	1	0

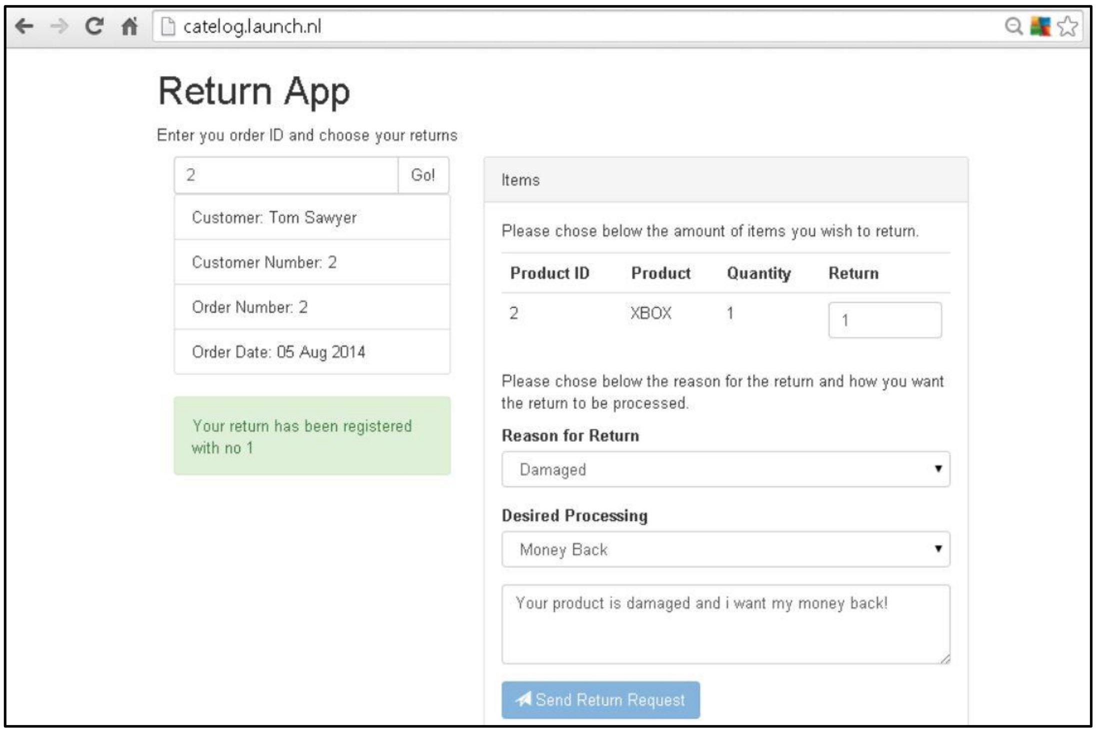
Please chose below the reason for the return and how you want the return to be processed.

Reason for Return: Damaged

Desired Processing: Money Back

Figure 18 Catelog front-end after customer enters order ID

After user clicks on the “Send Return Request” button, all the HTML form parameters will be first sent using HTTP POST method to the Heroku cloud database. The database will automatically generate a new return request record, indicated by a new Return ID as shown in the notification below the customer details in Figure 19. This Return ID will be then passed to the SaaS BPM tool along with all the other parameters using HTTP POST method for the approval process. Every return id in the database comprises of several attributes, out of which the return status ID is the most important. Initially this return status ID is set automatically to 1, which is “Pending for Approval”. Later on, this status will be adjusted depends on whether the return request is accepted or rejected.



Return App

Enter you order ID and choose your returns

2

Customer: Tom Sawyer

Customer Number: 2

Order Number: 2

Order Date: 05 Aug 2014

Your return has been registered with no 1

Items

Please chose below the amount of items you wish to return.

Product ID	Product	Quantity	Return
2	XBOX	1	1

Please chose below the reason for the return and how you want the return to be processed.

Reason for Return: Damaged

Desired Processing: Money Back

Your product is damaged and i want my money back!

Figure 19 Catelog front-end after return request has been submitted

In terms of the underlying technology, the dynamic front-end website was developed using HTML, Java Script and JQuery framework. For the layout, Bootstrap theme was implemented to make applying CSS (Cascading Style Sheet) styling easier. In addition to the client-side page, a simple admin page was developed also using the Boostrap Framework to facilitate system administrator in managing the database in an easier way. This page is shown in Appendix C – Heroku Admin Page Screenshot.

This web application interface has several limitations with respect to security. First, a customer should have been logged in before he can initiate a return request. In this prototype, this login feature is disregarded due to increased complexity. Nevertheless, since the main objective of this prototype is to show how disparate services can be integrated, rather than creating a perfect product, we can safely set the login feature aside for the time being. If the prototype is to be realized into a real commercial product, obviously this security aspect is something that is very crucial to be provided. Besides, exception handling strategies and form validation also have not been implemented yet.

5.4.2 Questetra Worflow (SaaS BPM)

HTML Form parameters that have been filled by the customer are then passed to a cloud-based BPM Tool namely Questetra. A BPM tool is adopted to support the synchronous process flow in our prototype, the approval process step, which requires human input to decide whether to approve the return request or not. Questetra BPM allows user to start a workflow remotely from outside of the BPM environment by using an HTTP request directed to that specific workflow. To identify which workflow to start, a number of parameters (processModelInfoId, nodeNumber, and more importantly API key of the user) have to be appended to the URL to which the HTTP Request will be posted.

Questetra prescribes the structure of the URL to be written as: https://s.questetra.net/<client_number>/System/Event/MessageStart/start. In this prototype, the URL is set as: https://shichijo-horikawa-319.questetra.net/System/Event/MessageStart/start?processModelInfoId=2&nodeNumber=0&key=[API Key]. The API Key is not revealed due to privacy concern. Both GET and POST method can be used, but in the case of a file type data is used or a long string needs to be passed to a parameter, the POST method is recommended.

The approval step process model is constructed using BPMN notation. The complete flow is depicted in Figure 20. The first node from the left is the HTTP Message Start Event. This component is responsible for starting the process triggered by an HTTP message received from outside of Questetra BPM Suite. The incoming message contains HTML Form Parameters that have been specified by the customer through the web application portal. These parameters are then used as Process Data Inputs to this approval process.

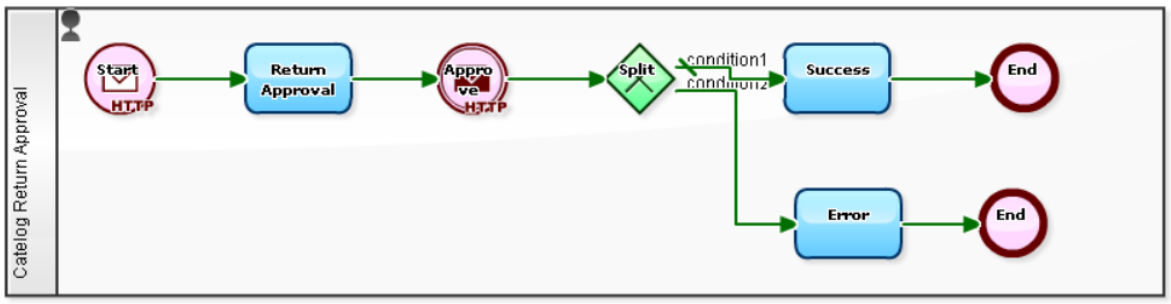
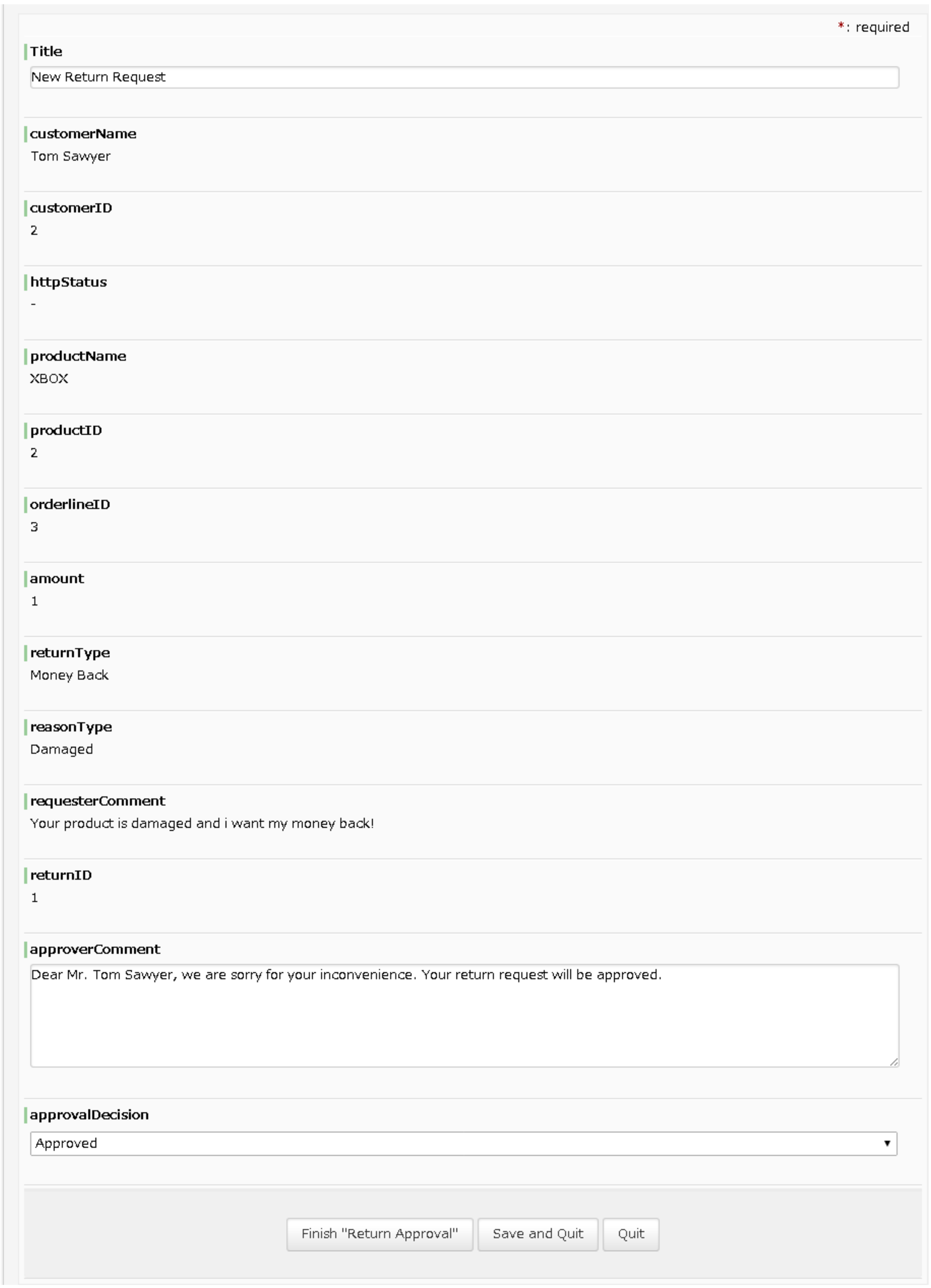


Figure 20 Return request approval workflow in Questetra

The second node is the actual approval step. Figure 21 displays the screen that the department in the E-commerce company who is responsible for handling return request will be able to see after a request has been received. All the HTML Form parameters that have been passed from the Catelog front-end webpage are displayed in the approval screen, starting from the customer name until the comment from the customer. The approver from the E-commerce company can then fill in the title of this request, provide comment and ultimately, decide whether to approve or reject the request.



*: required

Title
New Return Request

customerName
Tom Sawyer

customerID
2

httpStatus
-

productName
XBOX

productID
2

orderlineID
3

amount
1

returnType
Money Back

reasonType
Damaged

requesterComment
Your product is damaged and i want my money back!

returnID
1

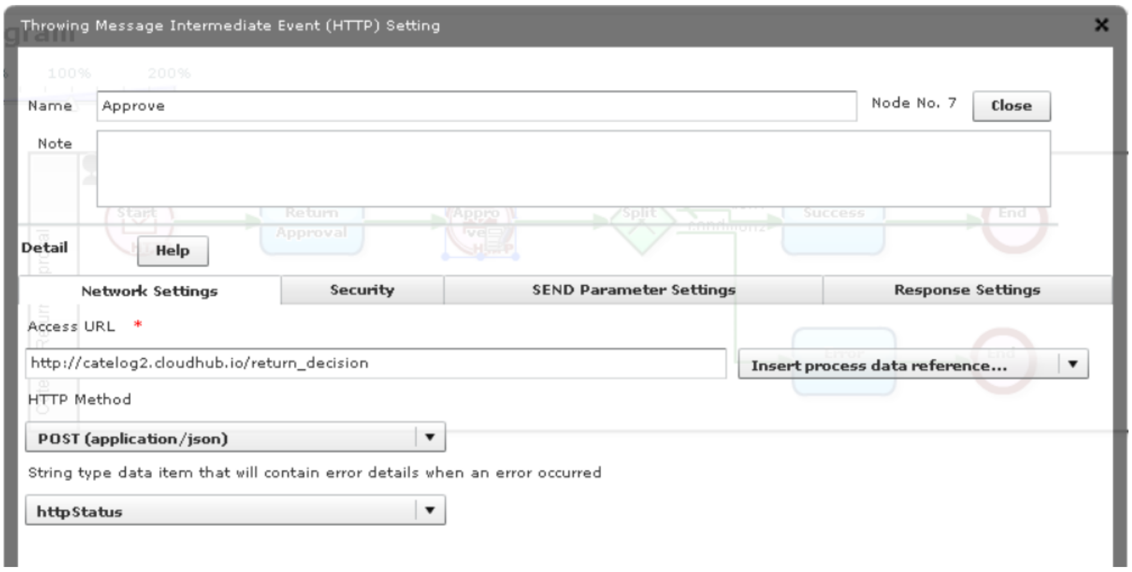
approverComment
Dear Mr. Tom Sawyer, we are sorry for your inconvenience. Your return request will be approved.

approvalDecision
Approved

Figure 21 Return request approval page in Questetra BPM

The third node is HTTP Throw Message Intermediate Event. In the middle of the execution of a process, this component can transmit messages containing Process Data to other process model or to external systems. The property screen in Figure 22 shows factors that can be adjusted in the HTTP Throw Message Setting menu. In the Network Setting tab shown in the figure, HTTP Request is sent to a specific external URL that will receive the message from Questetra. In this prototype the request is sent to Cloudhub, the cloud deployment platform of Mulesoft. POST method is used with content type application/json. HTTP status is used as string type data item to save the response of the process data as set in the Response Settings tab. If an error occurred, this data item will contain details of the error.

Furthermore, in the Security tab user can select to use either Basic authentication (with username and password of the external system that the message will be thrown to) or OAuth 2.0. In the SEND Parameter Settings tab user can specify which parameters to pass to the external URL, which in this case are the [Return ID], [Customer ID], [Approver Comment] and [Approval Decision]. The Approver Comment will be included in the email to be sent to the customer as an explanation why his return request is approved or rejected. The Approval Decision data item holds a value of “true” if the request is approved and “false” if rejected. Besides the aforementioned parameters, the Names of Process Data [Process ID], [Title] [Process ID] and [Title] parameters will be sent automatically.


Throwing Message Intermediate Event (HTTP) Setting

Name: Approve
Node No. 7
Note
Detail

Network Settings | Security | SEND Parameter Settings | Response Settings

Access URL *
http://catelog2.cloudhub.io/return_decision
HTTP Method
POST (application/json)
String type data item that will contain error details when an error occurred
httpStatus

Figure 22 HTTP Throw Message Intermediate Event property screen

If the entire operation can be completed successfully, the system will respond by “200” HTTP response code (OK). Otherwise, 500 response code will be returned, indication that error has occurred. In Figure 20, the entire flow will proceed to either Success or Error node based on the HTTP response code; 200 for Success and 500 for Error. This step is added solely to understand whether the process has been completed or not. After all process has been completed, all specified parameters will be passed back to Mulesoft as JSON object.

5.4.3 Mulesoft Studio Main Workflow

Figure 23 shows the main workflow in Mulesoft Studio. The first node in the workflow is an "HTTP Inbound" endpoint which is responsible for accepting incoming request from Questetra to the access URL: http://catelog2.cloudhub.io/return_decision. This endpoint is set to "request-response" mode because the response from the overall flow will be displayed in that specific URL. The incoming message with JSON format then needs to be transformed first because Mulesoft Studio does not have native supports for JSON format. Using the "JSON to Object" transformer, the JSON message is converted into Java HashMap format which is easier to handle by Mulesoft Studio. Specific keys and values are then extracted from the Hash Map for later use in the process flow. Initially, the customer ID is extracted from the original payload and then assigned to a Session Variable.

Mulesoft provides several types of variables: Flow Variable, Record Variable, and Session Variable. Since we want to be able to invoke the variable across multiple flows in the entire Mulesoft project, the Session Variable type is used. Besides the customer ID, the return ID and Approver Comment are also stored as Session Variable. The customer ID will be used to obtain customer details associated to that ID, the return ID will be used to change return status while the Approver Comment will be included in the email to be sent to the customer.

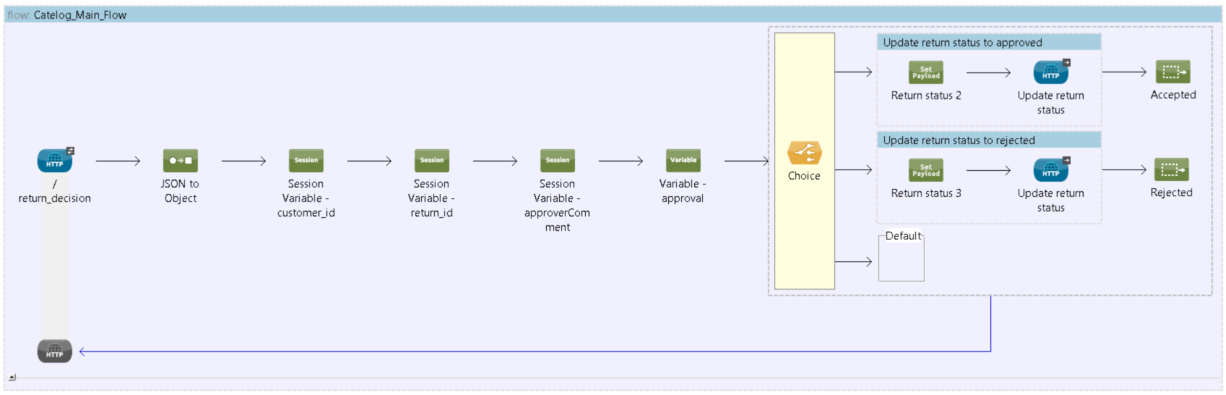


Figure 23 Mulesoft Studio main workflow

A Flow Variable then is assigned to the Approval Decision key of the message payload. Afterwards, a Choice Router will redirect the process flow according to the payload of the message that the flow is conveying. If the value of the Approval Decision in the Flow Variable is "true", which is the case when the return request is approved, the flow will be redirected to another flow namely "Accepted" (not visible in the figure above). On the other hand, if the payload holds a value of "false", the flow will be redirected into "Rejected" flow. If none of those conditions hold, the default flow will be selected.

The redirect process is executed through a flow reference component which is the rightmost component in the flow. Prior to redirecting the overall process flow, the return status ID for the corresponding return ID in the database has to be altered. A one-way HTTP Outbound endpoint is implemented to update the record in the Heroku Cloud Database. Before this endpoint, the message payload is specified using Set Payload transformer. At this point, the return status still holds value of "Pending Request Approval" with ID 1. If the return request is approved, the return status ID will be changed from 1 to 2 which is "request accepted" while if rejected, it will be changed into 3 which is "rejected".

5.4.4 Approved Request

Figure 24 displays the approved flow. The next sub-sections will explain each parts of the flow in details.

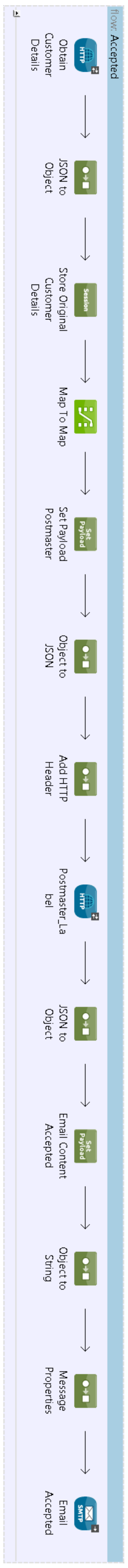


Figure 24 Accepted Flow Mulesoft

Obtain customer details from Database

As revealed in Figure 25, the first step in this flow is to obtain customer details from database based on customer ID of the requester. A cloud database which is based on PostgreSQL namely Heroku is used. Mulesoft Studio provides native support for database connection by using Database Connector. This connector allows users to link with almost any Java Database Connectivity (JDBC) relational database. As the name implies, JSBC is basically a Java-based API that enables users to execute operations over a data source system in consistent way using one same interface. Wide range of SQL operations and queries then can be executed on database using this API.

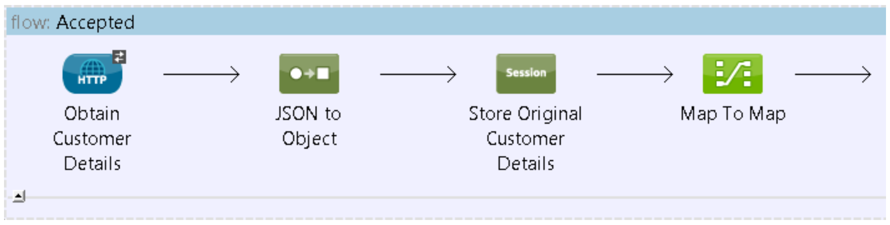


Figure 25 Obtain customer details from Heroku cloud database

In the platform design, rather than performing SQL Queries through the JDBC connector, RESTful API JSON Endpoints were developed from the existing database model using Python language with Flask Framework and Flask-Restless extension (Finkelstein, 2012). Through HTTP Requests, every collections (tables like customer, orders) and resources (specific instance of collection) in the database could be invoked. Table 3 below presents HTTP request methods and each of its corresponding database CRUD (Create, Read, Update, Delete) operations. Even though the REST endpoints took considerable amount of effort to develop upfront, it saved a huge amount of time and resources afterwards since it becomes much easier to perform database queries.

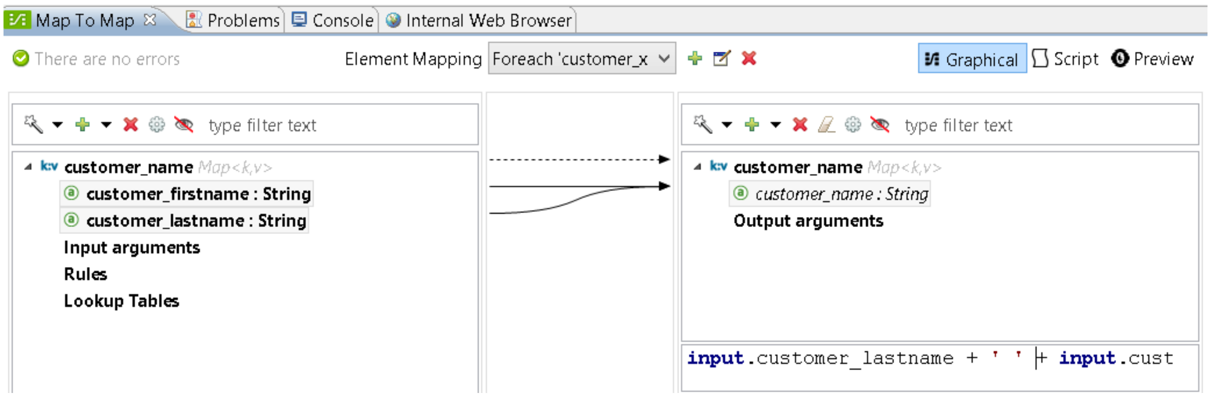
Table 3 CRUD actions and its corresponding HTTP Methods (Long Jump, 2014)

HTTP Methods	CRUD Action
POST	Create
GET	Read
PUT	Update
DELETE	Delete

All objects in the database are wrapped as REST endpoints, which are URL addressable resources. The URLs are constructed using this standard naming template: http://catelog.herokuapp.com/api/[collection]/[identifier]. To examine a specific data record, for instance product with ID 1, user then replace the [collection] part with "product" and the [identifier] part with "1" (without quotation marks). It is also possible to drill down to a specific attribute in each record. For example, to examine product category of product number 1, the request could be referenced to URL: http://catelog.herokuapp.com/api/product/1/product_category.

To fetch customer details, the REST call should be directed to http://catelog.herokuapp.com/api/customer/[customer_id]. The customer ID is assigned according to the ID of the customer placing the return request via Catelog web application portal. To illustrate, in this case the customer ID is 2, then the URL will be constructed as http://catelog.herokuapp.com/api/customer/2. As has been explained previously, the customer ID has been stored at the first place as a Session Variable. Consequently, the access URL should be specified as: http://catelog.herokuapp.com/api/customer/#[sessionVars.customerID] which will be built on-the-fly based on the customer ID.

After the customer details has been successfully fetched from the database, the returned JSON response needs to be transformed into Java Hash Map format because of the reason that has been mentioned in the previous section. Afterwards, the original message payload is stored as a session variable and then a schema mapping operation is executed using "Data Mapping" component. This mapping is necessary because in the database the customer name is stored in two different fields (customer_firstname and customer_lastname), but the shipping label service accepts customer name as the full name. Therefore, a simple mapping demonstrated in Figure 26 is designed to combine the first name and last name into a single customer name in the output field, separated by a single space. This customer_name field will be then placed in the payload to be sent to the shipping service.



```
input.customer_lastname + ' ' + input.cust
```

Figure 26 Schema mapping of customer name

Generate RMA (Return Merchandise Authorization) label by Postmaster

The customer details are needed as inputs in the RMA Label. To generate RMA label, Postmaster.io RESTful shipping service is used. Postmaster.io is a platform which enable developers to integrate parcel shipping and tracking into their existing systems using a common RESTful API. By aggregating data from various sources, the platform provides users with numerous shipping functionalities, for instance to compare shipping rates or time across different carriers (UPS, Fedex, or USPS), create shipping labels through its API or simply to validate a specific customer's address. As a result, shippers/businesses will be able to choose the fastest and most cost-effective way to ship a given parcel. Businesses can also flexibly specify file format of the label (PDF, PNG, or Vector) and label size.

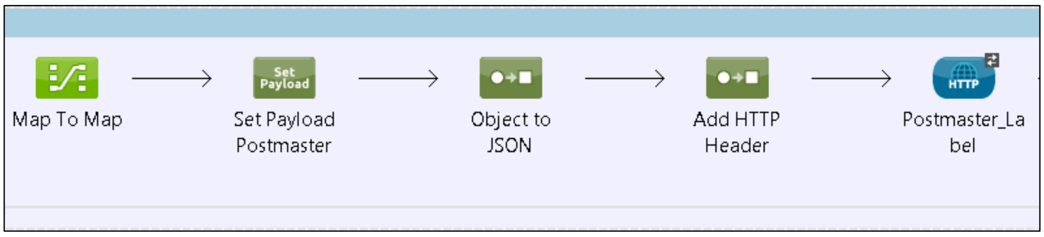
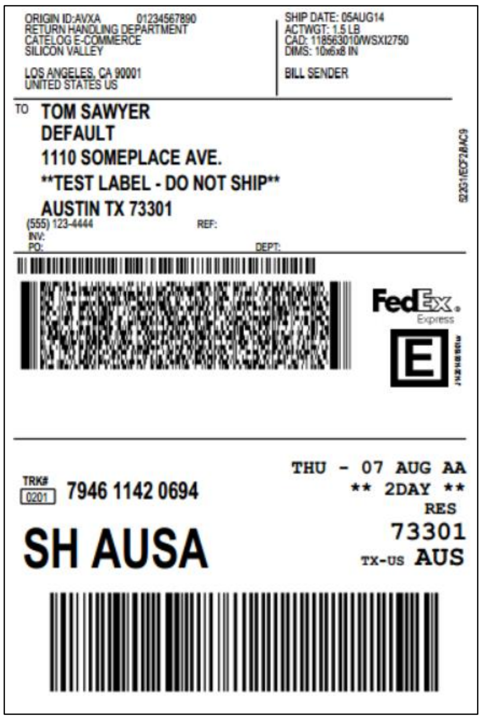


Figure 27 Shipping label generation workflow

In Figure 27, after the mapping operation a payload containing required information to be carried to the external shipping service is created using the "Set Payload" component. Inside the payload are information such as customer (address) information, carrier & service to choose, package dimension and label format (JPEG, PDF) & size. Furthermore, because initially this payload is still in a Java Hash Map format and the shipping service only accepts request in either XML or JSON format, an "Object to JSON" transformer needs to be placed to convert the payload back to JSON format. An HTTP header containing Content-Type and Authorization information is then assigned using a Message Properties Transformer component which is placed right before the HTTP Outbound endpoint. The Authorization header is generated automatically based on Basic Authentication mechanism. At the end, an HTTP POST request is directed towards https://api.postmaster.io/v1/shipments using the HTTP Outbound endpoint as the rightmost component in the figure.

If the request is success, Postmaster will return a response which encloses a URL to the shipping label that has been generated. In our design, instead of sending the file of the label itself to the customer, only the URL will be sent as a way to prevent the email being considered as a spam. The customer then could simply download the file, print the label, and then send the product to the suggested pick-up point address stated on the label in Figure 28.

Unfortunately, there is a little flaw in the label. The generated label is supposed to be an order fulfill-ment label (forward logistic direction), not a return label as currently being used. As a consequence, the label is specified as being sent from the E-commerce company (the top left part) to the customer (Tom Sawyer), not the other way around. For future development, this aspect needs to be taken into consideration. Aside from that, we can see that all customer details have been inscribed on the label. FedEx is chosen as the carrier, but since this label is generated in a sandbox environment, it can be observed that the label has been tagged as "Test Label – Do Not Ship" so the internal system of the carrier will be able to take correct actions on the label.



ORIGIN ID:AVXA 01234567890
RETURN HANDLING DEPARTMENT
CATELOG E-COMMERCE
SILICON VALLEY

LOS ANGELES, CA 90001
UNITED STATES US

SHIP DATE: 05AUG14
ACTWGT: 1.5 LB
CAD: 118563010/WSXI2750
DIMS: 10x6x8 IN

BILL SENDER

TO TOM SAWYER
DEFAULT
1110 SOMEPLACE AVE.
TEST LABEL - DO NOT SHIP
AUSTIN TX 73301
(555) 123-4444 REF:
INV:
PO: DEPT:

522G1/EOF2/8AC9

FedEx Express

E

TRK# 0201 7946 1142 0694

THU - 07 AUG AA
** 2DAY **
RES
73301
TX-US AUS

SH AUSA

Figure 28 Generated shipping label

Email to customer (Gmail)

The final step in the whole process is to send an email containing the URL link to the RMA label to the customer. As displayed in Figure 29 below, the response obtained from the Postmaster HTTP Outbound in the previous step is converted again from JSON to Java Hash Map. To create a customized content for the body of the email, a "Set Payload Transformer" is placed afterwards. The main content of the email is the "Approver Comment" parameter which has been stored as a Session Variable before. Additionally, customer name and most importantly the link to the RMA label are supplemented to the content. A "Message Properties" transformer arranges dynamic assignment to "To" field in the SMTP Endpoint according to the customer's email address. The sender of the email is set to a dummy email of Catelog E-commerce.

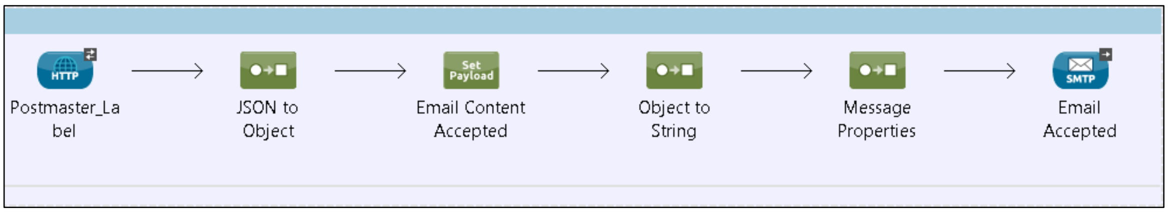


Figure 29 Email sending workflow

Last, an "SMTP (Simple Mail Transport Protocol) Transport" component is responsible to administer the sending of the email to the customer. It handles all of the basic email settings including SMTP host, port, email username, password, sender and receiver of the email. The content of the email is the message payload hitting the SMTP endpoint. Configuring SMTP settings for Google Mail (Gmail) service was a bit tricky because Gmail requires two-step authentication mechanism. User first needs to activate this security mechanism for his account and then generate an application-specific password for Mulesoft Studio. This password will be used as a substitute to the original password. TLS/SSL should also be enabled because Gmail service requires communication over secure protocol.

5.4.5 Rejected Request

If the return request from the customer is rejected, the customer will immediately receive a notification email informing that his request has been rejected along with the reason of rejection that has been given during approval step in Questetra BPM. Quite similar with the approval flow, in the Figure 30 first the customer details will be obtained through HTTP Outbound to Heroku REST Endpoints. Afterwards, the customer details will be stored as a Session Variable for easier invocation later. However, after this point there is no part where the request to generate a shipping label is performed. The rest of the flow is then exactly the same with the "Accepted Flow", concluded with sending email to the customer.

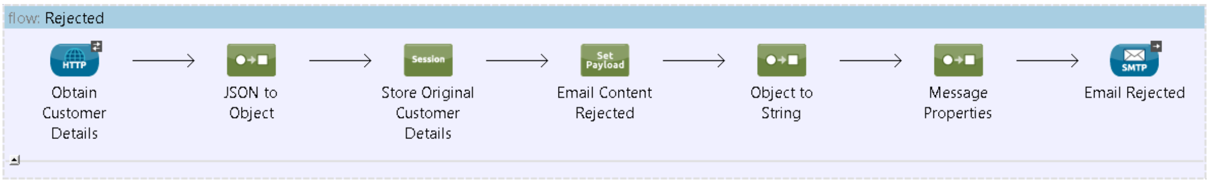


Figure 30 Rejected Flow Mulesoft

5.5 Design Validation

The E-commerce platform in this study is designed with the aim to promote pluggability of services as its main characteristic. Pluggability can be considered as a Non-Functional Requirement (NFR) of a system. To measure the extent to which the design has progressed towards this goal and it works as it is supposed to, a proper evaluation needs to be carried out. Nevertheless, because NFRs tend to have vague definition, performing an objective measurement of this type of requirement has been claimed as one of the most difficult activities (Subramanian & Chung, 2011).

Common understanding of what NFR is and how to evaluate the NFRs in the final software is generally difficult to achieve. Since pluggability is also a rather abstract concept, to measure how well our platform support pluggability of services, an approximation based on agility will be used. It seems logical to use agility as a surrogate measure for pluggability because, as stated in the chapter 1, one of the goals of our pluggable service platform is to increase enterprise agility. Organization can easily add or remove services to this platform to support its specific needs.

A considerable amount of literatures have been published with the topic on agility assessment in the context of Information Technology, or more specifically in software development processes. Lankhorst (2012), for instance, recognizes three main sources or enterprise agility: business agility, process agility and system agility. Business agility is rather high level because it is related to business strategy of an agile organization. Process agility also seems to be irrelevant in our case because we have defined beforehand the return handling business process model to be adhered. Therefore, system agility is chosen as the specific agility measure here. The following aspects have to be possessed by a system to be regarded as an agile system (Lankhorst, 2012):

1. The ease of making changes to a system: customizability, adaptability, analysability, changeability, scalability

2. The ease of rapidly deploying changes: learnability, installability, testability, manageability

3. The ease of minimizing and dealing with effects of changes: stability, fault tolerance, recoverability

4. The ease of integrating a system with its environment: interoperability and conformance to standards

5. The ease of decoupling a system from its environment: replaceability and reusability

Executing a quantitative measurement might be not feasible to do as it is not always easy to define proper metrics due to the characteristic of agility as NFR. Therefore, the evaluation of the prototype will be operated qualitatively instead. The developers as the one who understand the most the working mechanism behind this prototype are regarded to be eligible to perform the evaluation. The developers would be able to reflect back to their past experience in the whole development process of this prototype. The weakness of a qualitative assessment is obviously in its subjectivity, but that aspect will be minimized by making balance between the strong and weak points of this platform. Each of the five aspects above will be assessed in subsequent section.

To begin with, we tried to make changes to the system by completely replacing services that were consumed initially (i.e. Gmail as email service and Postmaster.io as shipping service) by another set of services. Figure 31 shows the original process flow to which changes will be operated. The Postmaster.io service invoked by the HTTP Outbound component (the blue node in the middle of the process flow) was replaced with Shipcloud.io. The same change was applied to the Google mail service shown by the rightmost node in the process model, which was replaced by Yahoo mail.

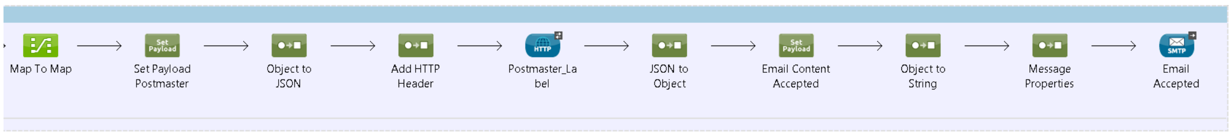


Figure 31 The original label generation and email sending process model

The overview of changes that needed to be applied is given in Table 4. First of all, the structure of the JSON Payload to be sent to the new shipping service needed to be tailored slightly to conform to the new service's required format. This adjustment was specified in the "Set Payload" transformer component (the second node from left). The next component that needed to be modified was the Set Message Properties transformer (shown as "Add HTTP Header"component in the flow). Here, the HTTP Authorization Header was modified from using Postmaster.io to the Shipcloud.io API Key.

Furthermore, the HTTP Outbound component was also adjusted by switching the original Postmaster destination URL (https://api.postmaster.io/v1/shipments) to the Shipcloud URL (https://api.shipcloud.io/v1/shipments), which is only slightly different. Because the format of the returned JSON response from Shipcloud is rather different than Postmaster, the value extraction step was also different. As can be seen in the table, the response from Postmaster for the label_url is not a complete URL but a text that needs to be appended to the end of "www.postmaster.io". Therefore, in the email to the customer, the label_url response can't be used directly. On the contrary, the returned label_url from Shipcloud is a fully-functional URL which can be sent directly to the customer's email.

Table 4 Illustration of changes from Postmaster to Shipcloud

Component	Postmaster	Shipcloud
JSON Payload	{ "to": { "company": "Catelog", "contact": "Mohammad Anggasta" "line1":"Emmastraat 210", "city": "Enschede", "state": "Null", "zip_code": "7513BH",	{ "to": { "company":"Catelog", "first_name":"Mohammad", "last_name":"Anggasta", "street":"Emmastraat", "street_no":"210", "country":"NL", "city":"Enschede", "zip_code":"7513BH" },

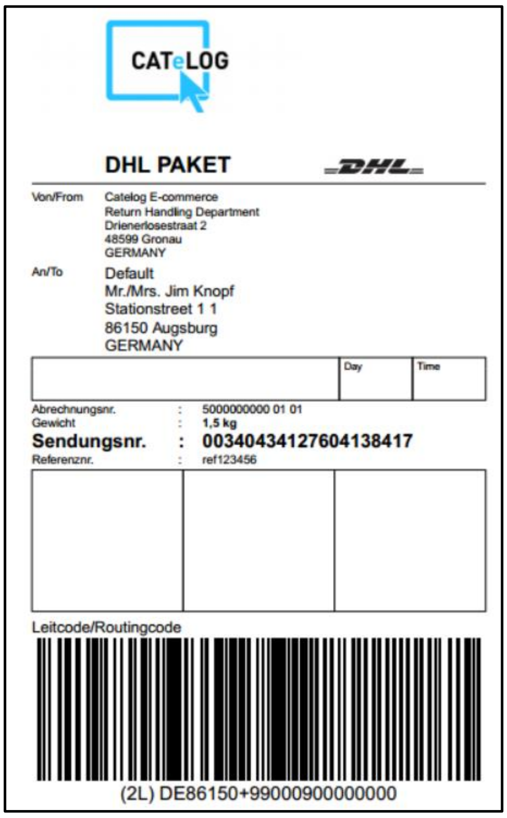
	"phone_no": "123456789" }, "package": { "weight": 1.5, "length": 10, "width": 6, "height": 8 } } "carrier": "Fedex", "service": "2DAY", "label": { "format":"PNG", "type":"NORMAL", "size":"SMALL"},	"package": { "weight":1.5, "length":20, "width":20, "height":20 }, "carrier":"DHL", "service":"standard", "reference_number":"ref123456", "create_shipping_label":true }
HTTP Header	Postmaster API Key	Shipcloud API Key
HTTP Outbound	https://api.postmaster.io/v1/shipments	https://api.shipcloud.io/v1/shipments
Returned Response (Label section)	"packages": [{ "weight_units": "LB", "weight": 1.5, "type": "CUSTOM", "height": 8, "width": 6, "length": 10, "label_url": "/v1/label/ AMIfv95bWF0VRo5Ioqoj7jdxENDL4tFUW2ejEACanLvxWZvNT9EZ-CQICDlcooyf2n7xzV3eBCuCgsK4rGImBQX-bOlDX2_e7gD_5adRgtLC7ueg3dIQooJrv31EpCYKApcZgS-6Lj1-s9IEdsl3vId-saIZDvyfKaKMmPW2UUDHBXo6kQQZ4kCiQ", "dimension_units": "Inches"	{ "id": "19dc5733fe5b025d01cc727f1b94a8ed8f0d5b39", "carrier_tracking_no": "00340434127604360535", "tracking_url": "https://track.shicloud.io/19dc5733fe", "label_url": "https://sc-labels.s3.amazonaws.com/ship-ments/70df0605/19dc5733fe/label/shipping_la-bel_19dc5733fe.pdf", "price": 5 }

Switching from the SMTP configuration of Gmail to Yahoo Mail was relatively straightforward as shown in Table 5. We merely had to change the hostname, port number, email address and password. Yahoo Mail does not require the two-step-authentication so the original account password could be directly used.

Table 5 Changing from Gmail to Yahoo Mail

Parameters	Gmail	Yahoo Mail
Host	smtp.gmail.com	smtp.mail.yahoo.com
Port	25	465
Password	Application-specific password (two-step authentication)	Original account password

Finally, all these changes resulted in the new shipping label as shown in Figure 32 below. This label was sent to the customer email's address using Yahoo Mail.


CATeLOG

DHL PAKET

Von/From Catelog E-commerce
Return Handling Department
Drienerlosestraat 2
48599 Gronau
GERMANY

An/To Default
Mr./Mrs. Jim Knopf
Stationstreet 1 1
86150 Augsburg
GERMANY

Day Time

Abrechnungsnr. : 5000000000 01 01
Gewicht : 1,5 kg
Sendungsnr. : 00340434127604138417
Referenznr. : ref123456

Leitcode/Routingcode

(2L) DE86150+99000900000000

Figure 32 Shipping Label generated by Shipcloud.io

By reflecting to this process of replacing the services initially used to completely different set of services, we could assess the extent to which our platform design support pluggability of services.

The ease of replacing existing services with the new ones

The pluggable service platform under study will operate mainly in the ever-changing, dynamic world of E-commerce. Accordingly, it should be easy to make changes to the platform to accommodate emerging business needs, customer requirements or latest development in the external environment that have not been considered in the initial design of the platform. Adaptability is then viewed as a highly important part of the software development lifecycle, which should be considered since the first step of software development, i.e. the software architecture design (Subramanian & Chung, 2001).

Extensibility is a tightly related concept to adaptability. A system can be called extensible if it allows external component to be plugged into the running system when needed. A more precise definition is given by Szyperski (1996) who argues that a system is considered as independently extensible if new extensions can be safely added without having to execute a global integrity check, or without having global knowledge of the entire system.

To replace the existing services with new ones, apparently only few components in the process model needed to be adjusted. As shown in Table 4 and Table 5, only slight configuration adjustments needed to be carried out. The nature of ESB architecture as the underlying design of the platform also makes it relatively simple to add or remove new services to the platform as only one application touch point/adapter is needed for each plugged in service/application. To make the data of the added component fit with the existing flow, a global integrity check is not necessary to be performed. As long as the newly added services/components can comply with the pre-defined canonical data model and schema, it can be ensured that they can work perfectly with the existing system. In that sense, changes to the workflow is isolated and integrity check could be accomplished locally. All aspects related to service lifecycle and governance have been dealt with by the Collaborative Service Platform component. Together with Collaborative Data Management component in the architecture which prescribes using schema mapping operation, a highly adaptable platform can be achieved.

Nevertheless, it should be realized that we didn’t try replacing REST services with SOAP Web Services. This change might require different adjustments in the platform design. Apart from that, adaptation capability also often corresponds with ability to make changes to business logic, not only easily but also automatically during run-time. Business logic provides guiding principle of how business objects interact with one another. In the domain of E-commerce, an adaptable web shop could, for example, automatically change the language of the web shop to fit with the country of the customer viewing the web shop.

It can be noticed from Figure 32 above that the customer was changed from customer number 2 (Tom Sawyer) to number 1 (Jim Knopf). The reason behind this change is that we wanted to tailor which service to use based on customer’s address. Because the customer 1 lives in Germany while customer 2 in USA, the new shipping service, Shipcloud, which is based in Germany is selected for customer 1 while Postmaster which is based in USA will be selected for customer 2.

In the current design of this prototype this change is still defined manually during design-time due to lack of Business Rules component. If a Business Rules engine is present, customized configurations can be applied to give unique experience for customers. The future development of this platform would be to create international shipping rules and fees. If the customer is located in Europe, Shipcloud will be automatically selected whereas for any other cases, Postmaster will be chosen.

The ease of rapidly deploying changes by the newly added services

Manageability is closely associated to maintainability that can be defined as a set of attributes with the goal to assess the ease with which a software system or component can be configured to adjust to a changed environment, correct faults, or improve performance (Candea, 2008). Manageability can be estimated by the time required to implement changes into the evaluated system. Once designed, our platform will be deployed to the cloud-based deployment platform namely Cloudhub. Based on our experience, deploying a compiled application to Cloudhub required in average around 3-4 minutes. Similarly, when we replaced the existing services with the new ones, it took some time to finish deployment of the changes. As a result, even though it only requires few minutes, the effect of changes couldn't be observed immediately.

The ease of minimizing and dealing with effects of the services replacement

Resilience can be defined as the ability to behave correctly despite unexpected changes in their business or execution environment. Resilience of a system is often interpreted into high availability, redundancy, error checking, exception handling or disaster recovery. It can be measured through the amount of disturbance or radical changes that the system under study can tolerate.

ESB architecture can facilitate addition, upgrade, replacement or removal of services with minimal interruption to existing environment because clients now send messages to the bus instead of interacting directly with target services. To test a new release of application before deploying it to the Production environment, a sandbox environment can be utilized. This can avoid unexpected consequences of changes to the system before the changes being deployed to the Production/Live environment. However, exception handling mechanisms are not yet incorporated in the current platform design. This might cause some issues when the existing services are replaced by new services, even though we did not encounter that during the development.

Apart from the resiliency in the platform design, Mulesoft Studio as the underlying integration platform also gives added value to resiliency. Being an Eclipse-based platform, Mulesoft has built-in error & consistency checking as well as exception handling component. It becomes easier for developers pinpoint the specific cause of error to then find solution to fix it. Besides, Mulesoft claims that they guarantee high availability and increased redundancy of Mule application instances, especially if deployed to Cloudhub.

The ease of integrating a system with its environment

One of the key determinants to achieve better interoperability is to conform to (widely adopted) standards while avoiding as much as possible the use of vendor-specific standards. Within SOA realm, it is suggested to use common & well-understood formats like XML and JSON for the message payload, or SOAP and REST for the web service interface. Specifically for REST, it has been widely considered in the industry that REST and HTTP as the best possible means to achieve intrinsic interoperability. The prototype is built on top of Mulesoft platform, which is Java-based and XML centric.

The preferred format in our platform design is JSON, which has the same openness characteristic and interoperability potential with XML. As can be seen from the process flow, the expected message payload and response from the REST services are always in the JSON format. The fully functional prototype itself also shows that interoperability among diverse applications and services has been achieved. The canonical data model in the Collaborative Data component was also designed by referring to the standard of E-commerce data model. This enables easier schema mapping from various sources to the reference model.

The ease of decoupling a system from its environment

SOA paradigm encourages service reusability to significantly improve application development productivity. Reusability is closely associated with modularity and service granularity. Modular decomposability concept from McGovern et al. (2003) divides an application into many smaller modules, with each module being responsible for a single function within the application. Besides application function, it is also possible to reuse specific parts of a business process in other business processes.

When carrying out SOA initiative enterprise architects are often suggested to carefully choose the right level of service granularity, which is the scope of functionality exposed by the service. Well-designed services could assist organization in achieving benefits such as more flexible business process and lower development cost. In determining the right granularity level, architects often have to make trade-offs between granularity and four aspects: performance, service genericity, service reusability and business process flexibility (Steghuis, 2006).

In the prototype we designed the REST API endpoints for each tables in the data model (customer, order, product, etc.) of the cloud database. REST API maps very well with the underlying data model of an organization's system. To query details of all customers, for instance, a GET operation could be performed to http://catelog.herokuapp.com/api/customer. To obtain details of specific resources (specific customer in this example), the service consumer can easily append the URL by adding the corresponding resources identifier (customer ID in this example) at the end of the URL like this : http://catelog.herokuapp.com/api/customer/1. By encapsulating database CRUD operations as REST Endpoints, it becomes highly reusable. In fact, during the development we could easily copy and paste all RESTful service building blocks from one process flow to the others.

The heavy use of RESTful services in this platform design can be viewed as the key to achieve reusability. REST promotes uniform and reusable service contract via common HTTP methods, which is something that SOAP WSDL services couldn't achieve. There is no need of separate documents (as WSDL in SOAP-based service) to describe resources of RESTful services. Fine-grained REST services make understanding and integration easier, but they tend to impact system performance due to a lot of requests generated to handle non-trivial use case. Coarser-grained REST API allows it to support a large number of known and unknown developers/users by handling API requests in a more generic way, not optimized for any of the requests. In general, designing services for reusability is not an easy task because sometimes it can't be foreseen what will be the future use of the services.

5.6 Discussion

In the previous sections, we have presented our methodology in the development of the prototype as well as the design validation afterwards. In this section, we will elaborate several discussion points based on our experience during the development of the prototype.

General Remarks

On a general note, while the use-case at hand offers limited complexity compared to a real world scenario and might not be able to be considered as a reference solution for practitioners, its goal is to evaluate the feasibility of the reference architecture presented in the previous section. The four external services used in the example correspond to four out of the five service types in the reference architecture as explained in chapter 4. While the integration of analytical services through file based integration is out of the scope, we were able to integrate the four other interface types into the collaborative flows.

We conclude that the cloud integration platform solutions currently only copy the capabilities of existing integration systems into the cloud without embracing the actual capabilities of a cloud based platform which is bringing different supply chain partners and their services together. To be more specific, if true multi-tenancy concept is applied in cloud-based integration platform, then it should be possible for business entities residing in that platform to collaborate with each other by granting access to relevant data and applications. While in the Customer-to-Customer market it is well understood that cloud based systems bring together different actors to collaborate and interact, this main advantage of cloud based systems seems not to be well understood in the organizational information systems field.

Collaborative Services and Process Framework

The collaborative service and process framework can be seen as a combination of existing Business Process Management (BPM), SOA Governance and API Management platforms. While the last two types of middleware have been adopted by cloud platform providers and integrated solution for service governance and process management exist, there is no cloud based solution available at the moment combining all the three concepts.

In the prototype we were able to implement the service based processes using the existing service bus solutions, but those are limited to short running synchronous scenarios. While it is possible to achieve service composition based on choreography, there is no possibility to have a central business process definition in form of an executable business process model. This turns large process flows difficult to analyze and maintain. While this is a typical limitation of an ESB like Mule, none of the investigated iPaaS services on the market supports the full set of workflow patterns or long running, complex processes executed by a workflow/BPM engine. There is also no support for standard process languages such as BPMN in existing cloud integration platforms. Consequently, we had to use a 3rd party cloud-based BPM tool to accommodate this need.

Furthermore, the support for inter-organizational interoperability is limited as the platforms target scenarios for integrating system owned by one single organization. While the architecture reflects inter-organizational scenarios by subscription and billing of services, the existing solutions focus only on technical aspects of integration. The aspect of technical interoperability between systems seems to be the main strong point of cloud based integration platforms. A large set of technology, application and cloud service connectors is available and the integration of systems on the technical level was quite straight forward while building the prototype. However, to achieve a full integration solutions, other aspects than the technical aspects should be brought into the picture such as financial, social, or security aspects.

Collaborative Data Management

The major drawback of existing cloud integration platforms is their generic business context. Our prototype has shown that the availability of shared data in the platform is crucial for flexible service integration. The collaborative data services are one key benefit of the platform when it comes to limit development effort to integrate services. Specifically, the use of canonical data model and schema mapping make the whole architecture more flexible and responsive to changes. Still, some aspects of collaborative data management are not covered in the prototype which will be crucial in a real world scenario. The existing backend systems of E-tailer like CRM or ERP systems have to be integrated with the collaborative resources in order to make the data available to the platform.

In addition, security mechanisms have to be implemented in order to define which business partner gets access to which data. This feature is not yet supported in the current version of the prototype. For further research, OAuth protocol could be implemented to enhance security aspect of the prototype. By using OAuth, business partners can exchange data and grant access to specific data to specific entities, which ensures an entity can only access part of data that is need for its specific operation.

Another important security aspect to take into account is the so-called CORS (Cross Origin Resource Sharing) Header. It is really crucial during development to enable this parameter in the external server that we want to pass data/parameters to. By default this feature is turned off to prevent unwanted external requests to the server. Therefore, at the beginning we always encountered errors every time we tried to send request from Catelog web app page to external servers. If the CORS feature is disabled, it is forbidden for a Javascript web page to send HTML Form Parameters using HTTP Post method. Fortunately, we have access to this external server,the Heroku Cloud Database. After the CORS parameter had been enabled, the submitted HTML form parameters could be successfully passed to the Heroku Cloud Database.

Service Classification

The classification of services and service interfaces in the architecture largely corresponds to the service types used in the prototype. Apart from data analytic services which are not applicable in the return registration case, the other four services types are present in the prototype.

The Postmaster RESTful shipping service can be considered as a generic standalone service which is highly reusable throughout different processes and business partners. This service also has a simple interaction pattern. Thus, providing as large set of standard technology adapters is crucial to integrate legacy services as well as services relying on specialized protocols.

Integrating SaaS into the process flow is one of the major barriers. The return application relies highly on business resources. Implementing the solution requires knowledge about the data services, implementing the web application is only possible with the collaborative data services in place. Overall the service classification scheme of the architecture can be considered as helpful, because each service type imposes different requirements on the platform.

On a side note, while most of the cloud integration platforms have a variety of connectors for common backend systems, connectivity to domain specific services of e-commerce are scarce. To achieve a high acceptance of the platform, services for all business functions in Chapter 3 have to be made available. Another finding of the prototype development is the fact that not all services are equally suited for cloud sourcing and loosely coupled integration.

Mulesoft flow, Message Structure, and Data Transformation

Some of the most crucial things to take into account to make the whole integration flow works is to know how Mulesoft integration flow should be constructed, how message is structured in Mulesoft, and what kind of data transformation to be executed. A flow in Mulesoft can be viewed as series of message-processing events. Each individual elements or building blocks within Mulesoft flow is responsible for handling the incoming message, processing, and routing. As shown in Figure 33, message object in Muesoft contains two main parts: message header and payload. The header itself contains metadata that consists of inbound and outbound properties of the message. The payload is the core of Mule message because it contains the business-specific data that user want to transport. Besides header and payload, Mule message might also contain variable, attachment and exception payloads.

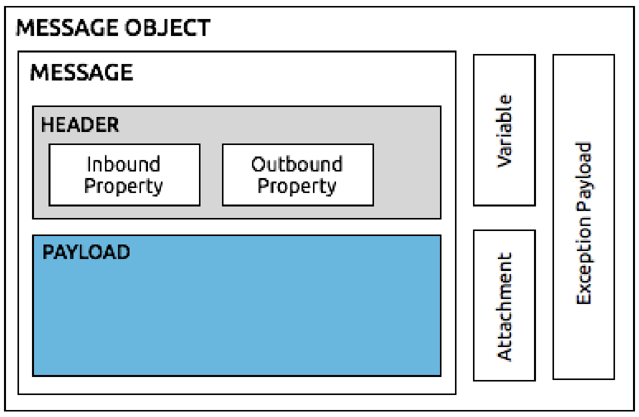


Figure 33 Mulesoft message structure (Mulesoft, 2014)

The developers had encountered many errors mostly because sometimes it wasn't fully understood what transformation to be executed on incoming message. In some cases when the value of the payload of a message needed to be extracted, error often occurred because of different data type. The workaround was to encapsulate the message payload as Flow Variable first before passing it throughout the integration flow. Selecting the most suitable MIME Type, or Content-Type header field, is also crucial to specify the nature of the data inside the body of the message. In our case, application/JSON is selected as the MIME type with UTF-8 encoding.

In general, working with Mulesoft still requires deep technical expertise. This tool might not be well-suited for use by business people even though it is also their concern to integrate different business partners together to complete business processes. A true pluggable platform might better to hide most complexities to users, like data transformation, by managing them by itself. As a result, a more business-friendly platform will enable business people and developers to collaborate in designing business process integration flows to meet organization's goals.

6. Conclusions

This research project was undertaken with the aim to create a reference architecture of a pluggable service platform for E-commerce. This final chapter concludes the findings of the preceding chapters by reflecting on the main research questions as well as each sub-questions, discussing contributions of this research, its limitations and finally, our recommendation for future research.

6.1 Answers to the Research Questions

The main question of this research as stated in the Chapter 1 serves as the main guideline in carrying out this project:

What reference architecture can best serve as the foundation for a pluggable e-commerce platform which supports seamless integration and coordination of e-commerce supply chain partners' application and services?

This main question is then divided into four broad sub-questions which are further operationalized into a number of more specific questions. Each chapter of this research, starting from chapter 2 to 5, has addressed each sub-questions. The first two sub-questions represent the theoretical part of this research and have been resolved by means of literature study and market analysis. The third question deals with design aspect of this research in which the reference architecture is constructed using the findings from sub-questions 1 and 2 by adhering to enterprise architecture modeling language and design principle that have been chosen. Lastly, the fourth question is answered through instantiaon of the reference architecture into a prototype product. A unique e-commerce case is selected, the prototype is produced through orchestration of real world services on top of an integration platform that has been chosen. The following section will provide thorough explanation for our findings in each chapter in order to answer each sub-question.

Research Question 1
What is the current e-commerce platform solutions landscape?

- What is the standard architecture of current e-commerce web shop platforms?
- What features/system components are generally provided? What features that might be necessary but the current platforms typically lack of?
- What issues are associated to the platforms with regards to integration and coordination?

This first sub-question sought to find out the state-of-the art of e-commerce web shop platform as the basis to derive a best of breed solution of e-commerce platform. With respect to e-commerce platform architecture, the findings of this study suggest that the most common approach in the market is to provide a number of features natively while at the same time allowing 3rd party services to be added into the platform. The difference from one web shop platform to the others justifies that some platforms might have more lightweight architecture than the others. The popularity of the platform is a huge factor to determine attractiveness of the platform for developers to create 3rd party services to that specific platform. For instance, Magento as one of the most popular platforms has a really extensive collection of 3rd party services offered in their own extensions marketplace.

At the end, choosing between one E-commerce platform to the other often comes back to size of the retailer, expected turnover, build & run budget. The proprietary platforms, which are generally Java-based, might be more suitable for middle to large retailers due to their maturity, extensive features, better enterprise support and integration with back-office systems. On the other hand, the open-source platforms, mostly PHP-based, have better ecosystem support, more flexibility and more budget-friendly. Besides differentiating e-commerce platforms based on its proprietary or open-source nature, differentiating them according to their deployment options (self-hosted or SaaS) also helped shed a light into differences in architecture design.

Afterwards, to discover common features of e-commerce platform both academic literatures and market sources are explored. Unfortunately, very little information could be found from the academic literatures. Most of the valuable information are obtained from the industry sources, through market reports or e-commerce platform vendor whitepapers. On a side note, this situation gives the impression that there are gaps between academic and practice within E-commerce domain which tend to be more industry-driven. Section 2.3 Features of E-commerce Web Shop Platform that have been compiled from various sources. Besides, features that might be essential but most of the platforms generally lack of have also been discovered such as return handling or multi-channel capabilities.

With respect to integration approach, in general it appears that the e-commerce platforms under study accomplish integration needs by relying on hard-wired web service based integration. In this approach, each external services is connected to each online shop platform through the so-called “connectors” or “adaptors”. This approach will unfortunately result in a spaghetti point-point architecture. This finding motivates us further to design our pluggable e-commerce platform architecture which relies on middleware architecture to handle the integration needs.

Research Question 2
What is the current integration platform solutions landscape?

- What is the role of integration in E-commerce domain?
- What is the standard architecture of current integration platforms?
- What features/system components are generally provided? What features that might be necessary but the current platforms typically lack of?

Chapter 3 of this report is directed towards answering this second sub-question. Chapter 3 starts by a comprehensive elaboration of the role of system integration in e-commerce value chain along with its history of development, from the legacy EDI system into the recent cloud-based integration platform. E-commerce is closely associated to Business to Business Integration (B2Bi) which aims to facilitate coordination and communication among the business partners, and Enterprise Application Integration (EAI) which deals with more internally-oriented integration needs. Integration tasks in E-commerce domain are challenging because of the diverse and distributed nature of systems in the enterprise network environment, especially for global companies with large number of supply chain partners. Proliferation of new technologies like cloud computing, mobile technology, social media also brings new challenges to the integration landscape. With proper design and implementation of integration, e-commerce companies could benefit from more streamlined business operation, lower transaction costs, dynamic business relationship and obtaining real-time information.

While the section above does not directly answer the question above, it gives a notion to readers of how we ended up with selecting cloud-based integration platform for this project. Subsequently, a state of the art analysis of cloud-based integration platform was carried out to discover standard architecture, features that are commonly provided and features that might be necessary but not generally provided. The results are then compiled into the Table 1 and Table 2, which contain list of meta-services architecture component. It is distinguished between “Service Framework” and “Process Framework”; the former is the architecture components to manage services throughout their lifecycle while the latter is responsible for handling integration process flows. These two frameworks comprise the main architectural component of the pluggable platform which is elaborated more in chapter 4.

Cloud-based integration platform vendors under study generally provide an Integrated Development Environment (IDE) to design and develop integration flows through a visual drag-and-drop user interface. Pre-built connectors are given along with the IDE, especially for common enterprise systems like SAP or popular cloud services like Salesforce. If a connector for a certain application/service is not already available in the basic repository, a connector software development kit (SDK) is generally proved to let users develop custom connectors. Most vendors also incorporate SOA Governance and API Management capabilities into their platform. Still, the platforms lack some features that might be crucial to deliver a complete solution to deal with various types of integration needs. For the most part, Business Process Management (BPM) Engine is not found in the platforms, making them unable to support complex, long-running business process which require human inputs to accomplish. We deal with this issue by using a 3rd party cloud-based BPM tool, invoked through REST API calls.

Research Question 3
How to design the reference architecture of pluggable E-commerce platform which support seamless integration and coordination?

- What architecture design principle and architecture modeling language to adhere?
- How should Business, Information System, and Technology domains of the platform be constructed?
- What components should be included in the architecture?

In this research, service-oriented is chosen in as a suitable design principle to create flexible E-commerce solutions. Service Oriented Architecture (SOA) principle promotes publishing of applications/systems as loosely-coupled services which can be invoked through open & standardized interface. If each of the functionalities of e-commerce platform are treated as a separate service, and if the core platform enables true pluggability of services, retailer could easily customize the platform according to their needs. Archimate language and TOGAF framework are chosen as the architecture modeling language and architecture framework to implement. These two standards, which are both developed by The Open Group, complement each other to make up a complete, powerful and integrated approach for delivering enterprise architecture. ArchiMate defines a well worked-out language, including a (graphical) notation to provide a concrete visualization for the architectures and views proposed in TOGAF.

To establish the complete reference architecture, separated parts are created first for each Business Actors that are involved a service-oriented e-commerce environment: the Online Retailer, the Integration Platform Provider and the Service Provider. The findings from Chapter 2 are mainly used as the basis to create the Online Retailer part, as well as for the “Collaborative Data Management” component design for the Platform Provider. The “Collaborative Data Management” component prescribes the creation of canonical data model and schema mapping as enablers of pluggable service platform.

Similarly, Chapter 3 serves as the foundation to derive the “Collaborative Service and Process Framework” component of the Platform Provider. This component contains the desired key functionalities and integration support requirements for the integration platform. Looking at the Service Provider role, several type of services that considered as relevant in the field of e-commerce have been identified. These services are published through various application interfaces. The “Collaborative Service and Process Framework” component is then responsible for managing the services throughout their lifecycle as well as integration process flows involving the services.

Research Question 4

How to implement and evaluate the reference architecture?

- What approach and tools to use to instantiate the architecture as prototype?
- Which parts of the e-commerce process to be selected as the case study of the prototype?
- How to evaluate the design of the architecture?

In order to select which integration platform to use to create the prototype, we refer back to market analysis that have been described in chapter 3. Out of the platforms under comparison, Mulesoft platform was selected due to its open source nature, freemium license, strong community support and suitability for our project. Mulesoft Studio has a steep learning curve. Even though graphical & model-driven approach is provided with the promise of lessen code required, developing applications using Mulesoft Studio might still require substantial technical (programming) skill & knowledge, in addition to good design capabilities.

Return registration/return handling is selected as the use-case for the prototype. This case is interesting to study not only because handling return requests properly could result in significant cost and time saving for retailer, but also because this feature is not supported by most of the e-commerce platforms basic packages. Return handling is mainly provided through 3rd party service providers.

After the prototype has been successfully created, it should be properly evaluated. However, because the nature of pluggability as Non-Functional Requirement which tend to have vague definition, performing a quantitative & objective measurement of this type of requirement has been claimed as one of the most difficult activities. To measure the extent to which the platform design support pluggability, agility is used as a surrogate measure. A number of literatures have proposed methods to assess agility aspect of a system. We referred to a research by Lankhorst (2012) which proposed five aspects of system agility to be assessed.

6.2 Contributions

This research brings noteworthy contribution to the academic world by adding insights to the growing body of literatures in the domain of e-commerce, application integration and enterprise architecture. Up to now, research addressing the topic of pluggable service platform, especially in e-commerce field, is very scarce. This study is expected to be able to fill this gap. We have compared a large number of academic and industry sources, analyzed them systematically in scientific manner, to then produce compilations of e-commerce web shop functionalities and integration platform architecture components. Besides enhancing our understanding of the aforementioned domains, the compilations serve as the basis to design the key building blocks of our reference architecture design. The reference architecture itself as the main artifact is the tangible forms of the contributions of this research. The design of our platform has taken into account the latest technologies development and their impacts on application integration landscape in e-commerce. Due to the novelty of this research, our architecture design might be still relevant in years to come.

In the context of practice, the main contribution of this research lies in the fully functional prototype itself. The findings from state-of-the-art analysis have shed a light on the latest advancement in the market of e-commerce web shop and integration platform solutions. This study has demonstrated the applicability of the reference architecture design to be instantiated as the prototype using variety of the most recent integration technologies and real world services in e-commerce domain, such as JQuery or REST API. Thus, with our architecture as the foundation the prototype could be developed further to become a real commercial product, for return handling or even to address wider range of e-commerce cases. Ultimately, the product may contribute as one of available solutions to the development of e-commerce field.

6.3 Limitation

As any other researches, this research embodies some limitations as well. The most notable limitation might be in prototype evaluation. The evaluation was performed qualitatively by the developers of the prototype, which makes it prone to subjectivity. A more proper evaluation would be to evaluate the prototype in more quantitative ways using well-defined metrics. The issue, however, was to find suitable metrics for Non-Functional Requirement like pluggability and agility.

Despite the novelty of this research from technical perspective, as an academic product the prototype does not embrace the complexity of the real world software product but rather focuses on specific aspects which might be unsuitable for a true commercial product. Some parts of the design have been simplified or taken as granted during the development of the prototype, for instance the security aspects. The collaborative service and process framework has only be partially evaluated. The process framework services for process development and operation are in place but monitoring capabilities are offered by the cloud integration platform only to the extent that technical measures are concerned. Business process monitoring and mining capabilities are not covered by state of the art iPaaS products.

6.4 Recommendation for Future Research

The architecture presented in this work covers a complete set of requirements for integration of external Services and APIs into an e-commerce landscape. Future work would be the discovery of other functional areas for innovation in the e-commerce process that can be implemented with the provided design. We also suggest a more detailed study on service classification with regards to their pluggability as the classification provided in the architecture can be only considered as a first draft towards a complete picture. It might be interesting to try incorporating different types of services into the platform to deal with new e-commerce scenarios that have not been addressed before. Cross-chain or cross-country-boundary e-commerce shipping could become interesting cases to implement the architecture design. If more than one case are implemented in future study, then the platform design could be investigated whether only specific cases of e-commerce are supported or various scenarios can be applied. This approach might be able to answer an open question about whether one architecture design fundamentally more agile than the others, or it depends on the selected case/business process scenarios.

With respect to prototype, the future work should incorporate security aspect into the design, for instance by adding OAuth protocol. OAuth can also support more dynamic collaboration of business partners as means of authentication. With OAuth, each actors in the e-commerce value chain would be granted access to specific data which they have the right to access. In addition, future work could be to test the usability of the final prototype by evaluating pluggability of the platform by using quantitative assessment. The methodology might be in the form of structured interview or survey by involving a sufficient numbers of potential users and experts from different background. Then, the feedbacks obtained from the validation sample could be used as inputs for further improvement of the prototype.

References

Almeida, J., Eck, P., & Iacob, M. (2006). Requirements Traceability and Transformation Conformance in Model-Driven Development. *2006 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, 355–366. doi:10.1109/EDOC.2006.45

Almulla, S. A., & Yeun, C. Y. (2010). Cloud Computing Security Management. In *Engineering Systems Management and Its Applications (ICESMA), 2010 Second International Conference on* (pp. 1–7).

Anteneh, A., Lertwachara, K., & Thongpapanl, N. (2010). Technology-Enabled Retail Services And Online Sales Performance. *Journal of Computer Information Systems*, *50*(3), 102–111.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., … Zaharia, M. (2010). A View of Cloud Computing. *Communications of the ACM*, *53*(4), 50–58.

Aulkemeier, F., Schramm, M., Iacob, M.-E., & van Hillegersberg, J. (n.d.). Towards a Service-Oriented Reference Model for the E-Tailing. *Unpublished*.

Barros, A., Dumas, M., & Hofstede, A. H. M. (2005). Service Interaction Patterns : Towards a Reference Framework for Service-Based Business Process Interconnection, (April), 1–26.

Bass, L., Clements, P., & Kazman, R. (2003). *Software Architecture in Practice* (p. 528). Addison-Wesley Professional. Retrieved from http://books.google.com/books?hl=en&lr=&id=mdiIu8Kk1WMC&pgis=1

Benesko, G. G. (1994). *Electronic commerce in the 21st century*. Retrieved from http://www.rtci.com/cent21.htm

Bernon, M., Rossi, S., & Cullen, J. (2011). Retail reverse logistics: a call and grounding framework for research. *International Journal of Physical Distribution & Logistics Management*, *41*(5), 484–510. doi:10.1108/09600031111138835

Bernstein, P. A., & Haas, L. M. (2008). Information Integration in the Enterprise. *Communications of the ACM*, *51*(9), 72–79.

Bertolino, A., & Polini, A. (2009). SOA Test Governance: Enabling Service Integration Testing across Organization and Technology Borders. *2009 International Conference on Software Testing, Verification, and Validation Workshops*, 277–286. doi:10.1109/ICSTW.2009.39

Candea, G. (2008). Toward Quantifying System Manageability. *HotDep*.

Chu, S.-C., Leung, L. C., Hui, Y. Van, & Cheung, W. (2007). Evolution of e-commerce Web sites: A conceptual framework and a longitudinal study. *Information & Management*, *44*(2), 154–164. doi:10.1016/j.im.2006.11.003

Costa, P. D., Pires, L. F., & Sinderen, M. Van. (2005). Designing a Configurable Services Platform for Mobile Context-Aware Applications. *International Journal of Pervasive Computing and Communications*, *1*(1), 13–24.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., & Weerawarana, S. (2002). Unraveling the Web Services Web - An Introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, *6*(2), 86–93.

Cusumano, M. (2010). Cloud computing and SaaS as new computing platforms. *Communications of the ACM*, *53*(4), 27. doi:10.1145/1721654.1721667

Czernicki, B. (2011). *IaaS, PaaS and SaaS Terms Clearly Explained and Defined*. Retrieved from http://www.silverlighthack.com/post/2011/02/27/iaas-paas-and-saas-terms-explained-and-defined.aspx

Denery, P. (n.d.). *Get Back*. Retrieved March 29, 2014, from http://www.internetretailer.com/2010/03/31/get-back

E-commerce Data Model. (2009). Retrieved from http://www.databaseanswers.org/data_models/e_commerce/index.htm

Ecommerce Europe. (2013). *Thuiswinkel.org: "Dutch online retail top sector within five years."* Retrieved April 08, 2014, from http://www.ecommerce-europe.eu/cms/showpage.aspx?id=498

E-commerce Europe. (2013). *Europe B2C Ecommerce Report 2013 Light Version* (p. 5). Brussel.

Endo, S., Yang, J., & Park, J. (2012). The investigation on dimensions of e-satisfaction for online shoes retailing. *Journal of Retailing and Consumer Services*, *19*(4), 398–405. doi:10.1016/j.jretconser.2012.03.011

Eroglu, S. A., Machleit, K. A., & Davis, L. M. (2001). Atmospheric qualities of online retailing - A conceptual model and implications. *Journal of Business Research*, *54*, 177–184.

Fensel, D., & Bussler, C. (2002). The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, *1*(2), 113–137. doi:10.1016/S1567-4223(02)00015-7

Fettke, P., & Loos, P. (Eds.). (2006). *Reference Modeling for Business Systems Analysis:* IGI Global. Retrieved from http://www.igi-global.com/chapter/perspectives-reference-modeling/28351

Finkelstein, J. (2012). *Flask Restless*. Retrieved from https://flask-restless.readthedocs.org/en/latest/

Foping, F., & Walsh, J. (2013). Design and Implementation of a Private RESTful API to Leverage the Power of an eCommerce Platform. In *15th International Conference on Information Integration and Web-Based Applications and Services, iiWAS 2013* (pp. 681–685). Vienna, Austria.

Genchev, S. E., Richey, R. G., & Gabler, C. B. (2011). Evaluating reverse logistics programs: a suggested process formalization. *International Journal of Logistics Management, The*, *22*(2), 242–263. doi:10.1108/09574091111156578

González, L., & Ruggia, R. (2010). Towards dynamic adaptation within an ESB-based service infrastructure layer. *Proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond - MONA '10*, 40–47. doi:10.1145/1929566.1929572

Hayashi, A. (1996). *Is corporate America ready for ecommerce?*

Hillegersberg, J. Van, Boeke, R., & Heuvel, W. Van Den. (2004). The Potential of Webservices to Enable Smart Business Networks. *Journal of Information Technology*, *19*(4), 281–287.

Hillegersberg, J. van, Moonen, H., & Dalmolen, S. (2012). Coordination as a Service to Enable Agile Business Networks. In J. Kotlarsky, I. Oshri, & L. P. Willcocks (Eds.), (pp. 164–174). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-33920-2_10

Huang, H., & City, S. (2011). The Study on E-tailing. In *2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce, AIMSEC* (pp. 1771–1774).

Humeau, P., & Jung, M. (2013). *Benchmark of e-Commerce solutions*. Retrieved from http://is.gd/e_commerce

IBM WebSphere Commerce. (2014).

IEEE. (2000). *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems* (p. 14). New York, New York, USA.

Jardim-Goncalves, R., Grilo, A., & Steiger-Garcao, A. (2006). Challenging the interoperability between computers in industry with MDA and SOA. *Computers in Industry*, *57*(8-9), 679–689. doi:10.1016/j.compind.2006.04.013

Jonkers, H., Lankhorst, M. M., ter Doest, H. W. L., Arbab, F., Bosma, H., & Wieringa, R. J. (2006). Enterprise architecture: Management tool and blueprint for the organisation. *Information Systems Frontiers*, *8*(2), 63–66. doi:10.1007/s10796-006-7970-2

Jula, A., Sundararajan, E., & Othman, Z. (2014). Cloud computing service composition: A systematic literature review. *Expert Systems with Applications*, *41*(8), 3809–3824. doi:10.1016/j.eswa.2013.12.017

Khoshnevis, S., Shams Aliee, F., & Jamshidi, P. (2009). Model driven approach to Service oriented Enterprise Architecture. *2009 IEEE Asia-Pacific Services Computing Conference (APSCC)*, 279–286. doi:10.1109/APSCC.2009.5394112

Kleeberg, M., Zirpins, C., & Kirchner, H. (2014). *Information Systems Integration in the Cloud: Scenarios, Challenges and Technology Trends*. (G. Brunetti, T. Feld, L. Heuser, J. Schnitter, & C. Webel, Eds.)*Future Business Software* (pp. 39–54). Cham: Springer International Publishing. doi:10.1007/978-3-319-04144-5

Kurz, C., Hotop, E., & Haring, G. (2001). Evaluation and Characterization of Business-to-Business Integration Systems, 424–438.

La Greca, P. (2014). *APIs: A New Path to SOA*. Retrieved from http://blogs.mulesoft.org/soa-and-api/

Lankhorst, M. (2005). Introduction to Enterprise Architecture. In *Enterprise Architecture at Work: Modelling, Communication And Analysis* (pp. 1–10).

Lankhorst, M. (2012). Agility. In M. Lankhorst (Ed.), *Agile Service Development: Combining Adaptive Methods and Flexible Solutions* (pp. 17–40). Springer Berlin Heidelberg.

Long Jump. (2014). *REST API*. Retrieved from http://lj.platformatyourservice.com/wiki/index.php?title=REST_API

Loser, C., Legner, C., & Gizanis, D. (2004). Master Data Management For Collaborative Service Processes. In *International Conference on Service Systems and Service Management, Research Center for Contemporary Management* (Vol. 2).

Maamar, Z. (2003). Commerce, E-Commerce, and M-Commerce: What Comes Next? *Communications of the ACM*, *46*(12), 251–257.

Magento Features List. (2014).

Maler, E., & Hammond, J. S. (2013, February). The Forrester Wave™: API Management Platforms, Q1 2013. Forrester.

Maleshkova, M., Pedrinaci, C., & Domingue, J. (2010). Investigating Web APIs on the World Wide Web. *2010 Eighth IEEE European Conference on Web Services*, 107–114. doi:10.1109/ECOWS.2010.9

Malinverno, P., Plummer, D. C., & Van Huizen, G. (2013, August). Gartner Magic Quadrant for Application Services Governance. Gartner.

Mellor, S. J., Scott, K., Uhl, A., & Weise, D. (2002). Model-Driven Architecture. In *Advances in Object-Oriented Information Systems* (pp. 290–297). Springer Berlin Heidelberg.

Miller, J., & Mukerji, J. (2003). *MDA Guide Version 1.0* (pp. 1–62).

Moharana, H. S., Murty, J. S., Senapati, S. K., & Khuntia, K. (2012). Coordination , Collaboration and Integration for Supply Chain Management. *International Journal of Interscience Management Review*, *2*(2), 46–50.

Mulesoft. (2013). *Increasing E-Commerce Agility* (p. 3). Mulesoft. Retrieved from http://www.mulesoft.com/lp/whitepaper/saas/ecommerce

Mulesoft. (2014). *Mule Message Structure*. Retrieved from http://www.mulesoft.org/documentation/display/current/Mule+Message+Structure

Nurmilaakso, J.-M. (2008). Adoption of e-business functions and migration from EDI-based to XML-based e-business frameworks in supply chain integration. *International Journal of Production Economics*, *113*(2), 721–733. doi:10.1016/j.ijpe.2007.11.001

Ouyang, C., Dumas, M., Ter Hofstede, A., & Van Der Aalst, W. (2006). From BPMN Process Models to BPEL Web Services. *2006 IEEE International Conference on Web Services (ICWS'06)*, 285–292. doi:10.1109/ICWS.2006.67

Papazoglou, M. P. (2003). Service -Oriented Computing : Concepts , Characteristics and Directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE'03)* (pp. 1–10).

Papazoglou, M. P., & Georgakapoulos, G. (2003). Introduction to the Special Issue about Service-Oriented Computing. *Communications of the ACM*, *46*(10), 24–29.

Papazoglou, M. P., & Heuvel, W.-J. (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, *16*(3), 389–415. doi:10.1007/s00778-007-0044-3

Pasley, J. (2005). How BPEL and SOA Are Changing Web Services Development. *Internet Computing*, *9*(3), 60–67.

Pautasso, C., & Leymann, F. (2008). RESTful Web Services vs . " Big " Web Services : Making the Right Architectural Decision Categories and Subject Descriptors. In *Proceedings of the 17th international conference on World Wide Web* (pp. 805–814). ACM.

Peffers, K., Tuunanen, T., Rothenberger, M. a., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77. doi:10.2753/MIS0742-1222240302

Pezzini, M. (2011). *Integration PaaS : Enabling the Global Integrated Enterprise*. Las Vegas.

Pezzini, M., & Lheureux, B. J. (2011). *Integration Platform as a Service : Moving Integration to the Cloud*.

Pezzini, M., Natis, Y. V, Malinverno, P., Iijima, K., Thompson, J., & Thoo, E. (2014). Magic Quadrant for Enterprise Integration Platform as a Service Market Definition / Description, (January).

Potočnik, M., & Juric, M. B. (2012). Integration of SaaS using IPaaS. In *Proceedings of the 1st International Conference on CLoud Assisted ServiceS* (pp. 35–41). Bled.

Power, D. (2005). Supply chain management integration and implementation: a literature review. *Supply Chain Management: An International Journal*, *10*(4), 252–263. doi:10.1108/13598540510612721

Pyke, D., Johnson, M. E., & Desmond, P. (2001, January). E-fulfillment: Its Harder Than It Looks. *Supply Chain Management Review*, 26–32.

Rao, S., Griffis, S. E., & Goldsby, T. J. (2011). Failure to deliver? Linking online order fulfillment glitches with future purchase behavior. *Journal of Operations Management*, *29*(7-8), 692–703. doi:10.1016/j.jom.2011.04.001

Research In Action. (2012). *The Hidden Costs of Managing Applications in the Cloud* (p. 4). Research In Action GmbH.

Ricker, F. R., & Kalakota, R. (1999). Order Fulfillment : The Hidden Key to e-Commerce Success. *Supply Chain Management Review*, *11*(3), 60–70.

Ried, S. (2014, February). The Forrester Wave^{TM}: Hybrid Integration, Q1 2014. Forrester.

Rimal, B. P., Jukan, A., Katsaros, D., & Goeleven, Y. (2010). Architectural Requirements for Cloud Computing Systems: An Enterprise Cloud Approach. *Journal of Grid Computing*, *9*(1), 3–26. doi:10.1007/s10723-010-9171-y

Samtani, G. (2002). *B2B Integration*. (M. Healey & S. Samtani, Eds.) (p. 98). London: Imperial College Press. doi:10.1142/p263

Schepers, T. G. J., Iacob, M. E., & Van Eck, P. a. T. (2008). A lifecycle approach to SOA governance. *Proceedings of the 2008 ACM Symposium on Applied Computing - SAC '08*, 1055. doi:10.1145/1363686.1363932

Sengar, P., Alvarez, G., & Fletcher, C. (2013). *Magic Quadrant for E-Commerce*.

Sikander, J., & Sarma, V. (2010). *A Prescriptive Architecture for Electronic Commerce and Digital Marketing* (p. 21).

SOA. (2010). Retrieved from http://stlarch.blogspot.nl/2010/11/soa.html

Steghuis, C. (2006). *Service Granularity in SOA Projects: A Trade-off Analysis*. University of Twente.

Streekmann, N., Steffens, U., Claus, M., & Garbe, H. (2006). Model-Driven Integration of Business Information Systems. *Softwaretechnik-Trends*, *26*(4), 1–5.

Su, C.-J., & Chiang, C.-Y. (2012). Enabling successful Collaboration 2.0: A REST-based Web Service and Web 2.0 technology oriented information platform for collaborative product development. *Computers in Industry*, *63*(9), 948–959. doi:10.1016/j.compind.2012.08.018

Subramanian, N., & Chung, L. (2001). Software Architecture Adaptability : An NFR Approach. In *Proceedings of the 4th International Workshop on Principles of Software Evolution* (pp. 52–61). ACM.

Subramanian, N., & Chung, L. (2011). Metrics for Software Adaptability. In *Proceedings of Software Quality Management 2011*.

Szyperski, C. (1996). Independently Extensible Systems - Software Engineering Potential and Challenges. *Australian Computer Science Communications*, *18*, 203–212.

Tang, A., Han, J., & Chen, P. (2004). A Comparative Analysis of Architecture Frameworks. In *Software Engineering Conference, 2004. 11th Asia-Pacific* (pp. 1–15). IEEE.

The Open Group. (2013). *Archimate 2.1 Specification* (p. 5). Berkshire.

TOGAF. (2009). *The Open Group Architecture Framework, Version 9, Enterprise Edition*. Retrieved from TOGAF (2009). The open group architecture framework, version 9, enterprise edition. Document number: G091. Retrieved Novem- ber 29, 2010, from The Open Group: http://www.opengroup.org/ architecture/togaf9-doc/arch/.

Touzi, J., Benaben, F., Pingaud, H., & Lorré, J. P. (2009). A model-driven approach for collaborative service-oriented architecture design. *International Journal of Production Economics*, *121*(1), 5–20. doi:10.1016/j.ijpe.2008.09.019

Trade it ™ Ecommerce Platform Feature List. (2014).

Turban, E., Lee, J. K., King, D., McKay, J., & Marshall, P. (2007). Building e-commerce applications and infrastructure. In *Electronic Commerce : A Managerial Perspective* (1st ed., pp. 19–1 – 19–53). Prentice Hall.

Urbaczewski, L., & Mrdalj, S. (2006). A comparison of enterprise architecture frameworks. *Issues in Information Systems*, *VII*(2), 18–23.

Van Hillegersberg, J., Zuidwijk, R., van Nunen, J., & van Eijk, D. (2001). Supporting Return Flows in the Supply Chain. *Communications of the ACM*, *44*(6), 74–79.

W3C. (2004). *Web Services Glossary*. Retrieved from http://www.w3.org/TR/ws-gloss/

Walker, B. K. (2012). *The Forrester Wave TM : B2C Commerce Suites*.

Weill, P., & Vitale, M. (2013). *Place to Space: Migrating to Ebusiness Models (Google eBook)* (p. 372). Harvard Business Press. Retrieved from http://books.google.com/books?hl=en&lr=&id=uHv5gvXt2NUC&pgis=1

Wick, M., Rohanimanesh, K., Schultz, K., & Mccallum, A. (2008). A Unified Approach for Schema Matching, Coreference and Canonicalization. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 722–730). ACM.

Ystats.com. (2013). *Global B2C Ecommerce Market Report 2013* (Vol. 49). Retrieved from http://www.ystats.com/uploads/report_abstracts/1021.pdf

Zachman, J. a. (1987). A framework for information systems architecture. *IBM Systems Journal*, *26*(3), 276–292. doi:10.1147/sj.263.0276

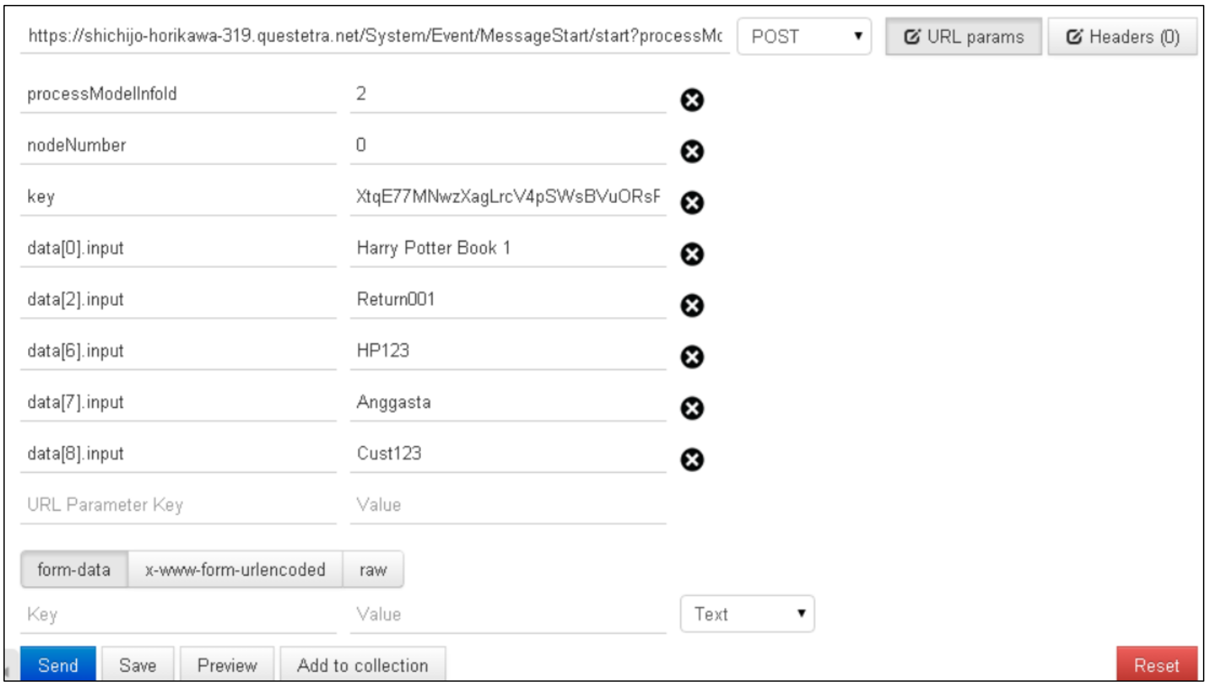
Zhonghua, D., & Erfeng, H. (2010). Analysis of SaaS-Based E-Commerce Platform. *2010 International Conference on E-Business and E-Government*, 9–12. doi:10.1109/ICEE.2010.10

Zur Muehlen, M., Nickerson, J. V., & Swenson, K. D. (2005). Developing web services choreography standards—the case of REST vs. SOAP. *Decision Support Systems*, *40*(1), 9–29. doi:10.1016/j.dss.2004.04.008

Appendix A – Postman HTTP Client

During the prototype development, in order to immediately test and debug the effect of changes in Mulesoft flow configuration, the Mulesoft project was deployed locally on Mulesoft built-in server. This approach significantly saved development time (only 10 seconds) compared to deploying the Mulesoft project on Cloudhub, which could take in 5 minutes in average. Then, Postman HTTP Client was used to send HTTP request to the deployed application as a replacement to Questetra BPM.

Postman is a powerful HTTP Client which is typically used to test RESTful Services. User can easily build any HTTP request by using Postman's intuitive and clean user interface. As shown in Figure 34, uers can specify URI to call, authentication method and HTTP method to use, header to put and parameters to carry. Basic, Digest and OAuth 1.0 authentication method are supported. Users are also able to store specific HTTP requests as Collection for later use. This feature is really helpful in testing Web Services. However, at the moment Postman is only provided as Google Chrome browser extension, which might limit its flexibility.



https://shichijo-horikawa-319.questetra.net/System/Event/MessageStart/start?processMc

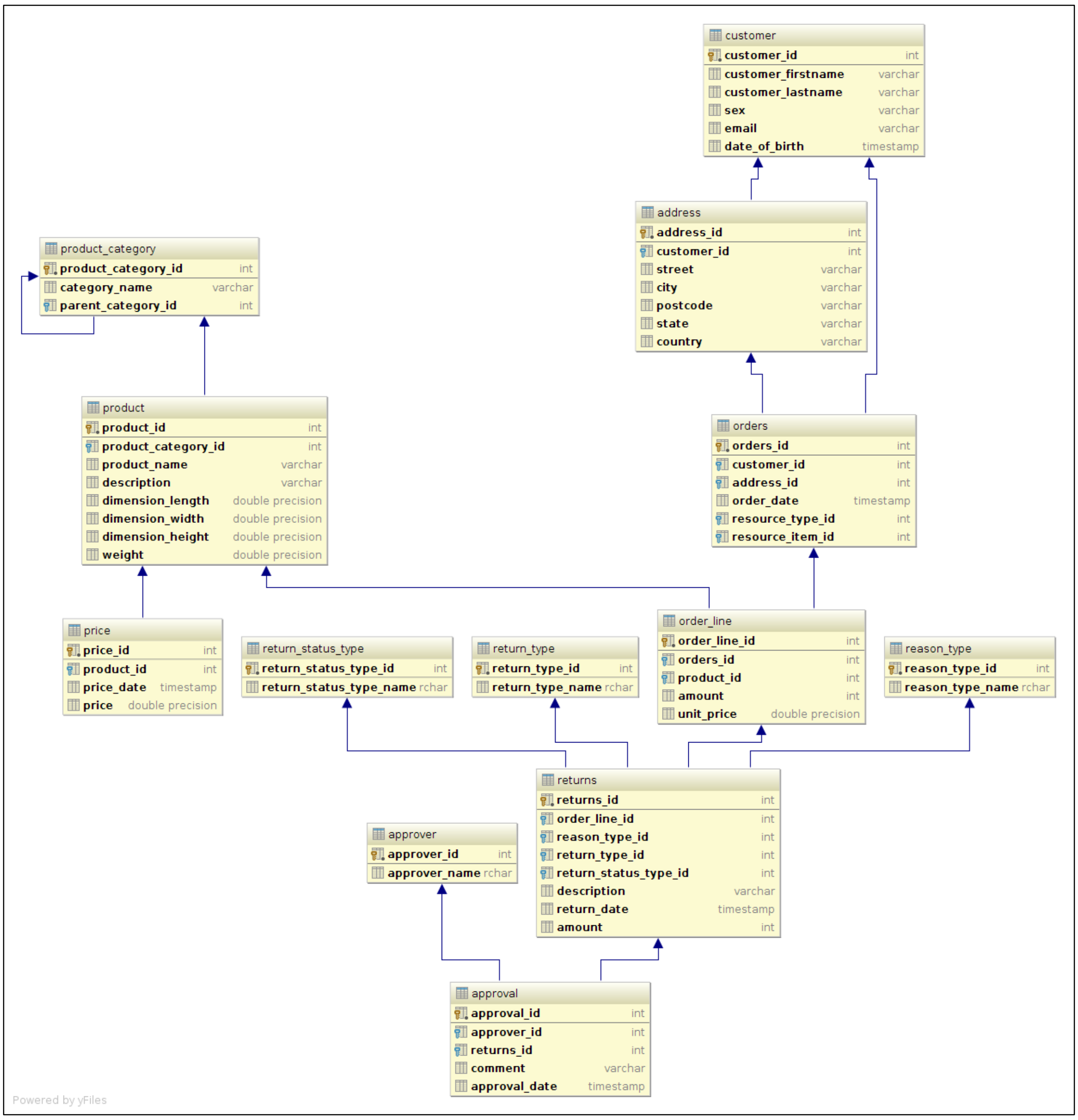
POST

processModelInfoId	2
nodeNumber	0
key	XtqE77MNwzXagLrcV4pSWsBVuORsF
data[0].input	Harry Potter Book 1
data[2].input	Return001
data[6].input	HP123
data[7].input	Anggasta
data[8].input	Cust123

Figure 34 Postman Interface filled with our sample HTTP request

Appendix B – Heroku Database Schema

The Figure 35 shows the data model of our Heroku cloud database. This model serves as the canonical data model which should be referred by services that are consumed by our platform. The schema mapper tool will aid the transformation from the source data model into this data model.



customer	
customer_id	int
customer_firstname	varchar
customer_lastname	varchar
sex	varchar
email	varchar
date_of_birth	timestamp

address	
address_id	int
customer_id	int
street	varchar
city	varchar
postcode	varchar
state	varchar
country	varchar

product_category	
product_category_id	int
category_name	varchar
parent_category_id	int

product	
product_id	int
product_category_id	int
product_name	varchar
description	varchar
dimension_length	double precision
dimension_width	double precision
dimension_height	double precision
weight	double precision

orders	
orders_id	int
customer_id	int
address_id	int
order_date	timestamp
resource_type_id	int
resource_item_id	int

price	
price_id	int
product_id	int
price_date	timestamp
price	double precision

return_status_type	
return_status_type_id	int
return_status_type_name	rchar

return_type	
return_type_id	int
return_type_name	rchar

order_line	
order_line_id	int
orders_id	int
product_id	int
amount	int
unit_price	double precision

reason_type	
reason_type_id	int
reason_type_name	rchar

returns	
returns_id	int
order_line_id	int
reason_type_id	int
return_type_id	int
return_status_type_id	int
description	varchar
return_date	timestamp
amount	int

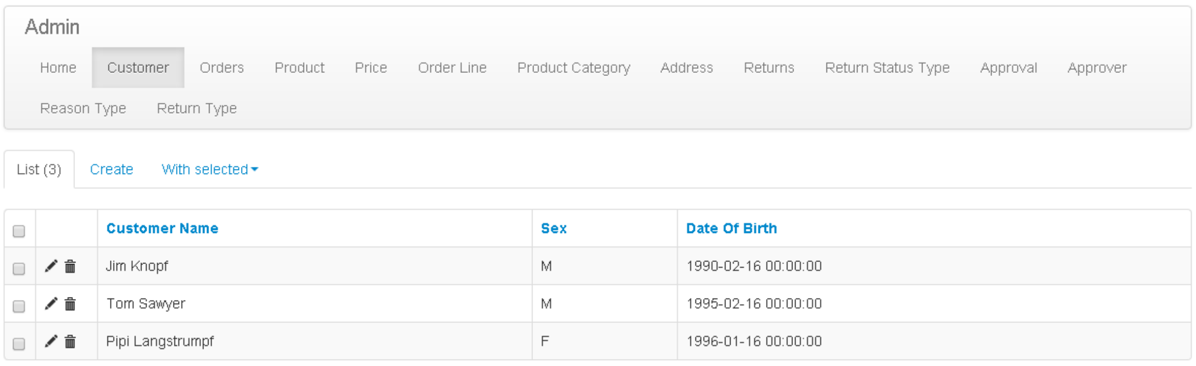
approver	
approver_id	int
approver_name	rchar

approval	
approval_id	int
approver_id	int
returns_id	int
comment	varchar
approval_date	timestamp

Figure 35 Heroku database data model

Appendix C – Heroku Admin Page Screenshot

As part of Collaborative Data Management initiative, we created an Admin portal as shown in Figure 36 to manage the shared Heroku cloud database which can be accessed by anyone who has the right credentials /authorized parties. However, the current version does not employ any authentication or authorization mechanism yet.



Customer Name	Sex	Date Of Birth
Jim Knopf	M	1990-02-16 00:00:00
Tom Sawyer	M	1995-02-16 00:00:00
Pipi Langstrumpf	F	1996-01-16 00:00:00

Figure 36 Heroku Admin Page