Integrating Order Management System Simulator in HCL Commerce

Ákos Mándi

Bachelor’s thesis
May 2022
Information and Communication Technologies
Degree Programme in Information and Communications Technology

Description

Ákos Mándi

Integrating Order Management System Simulator in HCL Commerce

Jyväskylä: JAMK University of Applied Sciences, May 2022, 53 pages

Information and communication technologies. Degree Programme in Information and Communications Technology. Bachelor’s thesis.

Permission for open access publication: Yes

Language of publication: English

Abstract

The author of this thesis participated in an internship at Solteq Oyj during his Erasmus studies in Finland, where he was looking for a solution to a real development problem in HCL Commerce (formerly WebSphere Commerce). The paper describes the popular eCommerce platforms of today and the role of an Order Management System, which is vital for the operation of an eCommerce system. In addition to these, the main focus is on integrating the OMS Simulator system provided by HCL in the project development environment, solving compatibility issues. Java, Spring Boot, and Apache Camel technologies are also mentioned in the solutions and their practical use and results are presented in this thesis. Besides presenting the communication between the eCommerce project systems, the messages and their states are also discussed in the paper.

At Solteq, HCL Commerce software is used extensively every day while developing eCommerce solutions for customers. There was a need to find an answer to the following question: how an order management system could be simulated during development, and how could it be integrated into the existing systems while also discovering the possible compatibility issues. The task was to find possible solutions for this question and overcome the challenges by searching for different methods and techniques to solve the communication between the OMS Simulator and the existing eCommerce site. The final result is a thesis presenting several solutions that can be a solid basis for future system integration in the project development environment.

Keywords/tags (subjects)

eCommerce, HCL commerce, WebSphere Commerce, Order Management System

Miscellaneous (Confidential information)

None

Contents

Figures

Tables

Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
BI	Business Intelligence
DOM	Distributed Order Management
EAR	Enterprise Archive
eCommerce	Electronic Commerce
EJB	Enterprise Java Bean
HTTP	HyperText Transfer Protocol
JAR	Java Archive
JMS	Java Message Service
JSP	Java Server Pages
JVM	Java Virtual Machine
OMS	Order Management System
PaaS	Platform-as-a-service
POM	Project Object Model
RAT	Runtime Analysis Tools
RAD	Rational Application Developer
REST	Representational State Transfer
SaaS	Software-as-a-service
SKU	Stock Keeping Unit
SOAP	Simple Object Access Protocol
UI	User Interface

URL	Uniform Resource Identifier
WSDL	Web Services Description Language
XML	Extensible Markup Language

1 Introduction

1.1 Background of the work

The author's employer, Solteq Oyj provided the basic concept and research topic for this work. Until July 2022, the author is completing his practical training as a Software Trainee in the Omni commerce team. This project helped the author to understand and get familiar with the fundamental ideas of eCommerce systems, particularly HCL commerce, as well as the day-to-day development duties of a webstore containing thousands of items to offer. While researching this topic, the author learned about the architecture and dataflow of an existing commerce project while also analyzing and assessing the possible methods of improving the development environment.

1.2 Objectives

The project's main aim was to find answers to how an Order Management System could be simulated during the development and testing steps of an eCommerce webstore for one of the clients of Solteq and set up the OMS Simulator server in the development environment. During developing and testing a website, it is essential to simulate the behaviour of a Distributed Order Management (DOM) system with which we can create inventory levels and use them in the storefront to show stock availability. The external OMS Simulator serves as a placeholder for a functional DOM system, where generic inventory levels are generated and used to integrate into the storefront stock availability features. Without this, the ordering process cannot be thoroughly tested as a new feature can contain possible bugs which would be discovered only after the solution is used inside the test or production environment. This could prolong the development and testing phase of releasing a new feature and could cause a disruption in the day-to-day operation of the client's business. With the integration of an OMS Simulator, this risk can be easily mitigated while creating more stable and well-tested solutions.

Before answering these questions, it is essential to understand the basic architecture and technical building blocks of the eCommerce site, in our case a site that is based on Version 8 of HCL WebSphere Commerce software. This work also addresses the concepts of Order Management Systems while showing the communications and messages between an OMS Simulator and a functional website. The main objective is to establish and introduce the possible options for solving compatibility issues between these two while also making the reader familiar with the theory and dataflow behind these systems.

1.3 Research methodology

During this work, the main aim was to provide solutions for an issue while researching the available technologies and best practices in the industry. The implementation method of this project was the development of a service or product as the primary goal was to provide software solutions for integrating an OMS Simulator to an existing eCommerce project's development environment while solving compatibility issues.

During the first phase of the project, the initial issue and technologies around it were investigated while looking at existing solutions that could be used as possible candidates to solve the compatibility issues between the OMS Simulator and the existing site in the development environment. The next phase was to define the solutions and the technologies which are used to implement them while also making a piece of software that can be a promising starting point to continue the implementation after the development team at Solteq chooses the best solution.

1.4 Solteq

Solteq is a Nordic IT services and software solutions company specializing in digitizing business and industry-specific software. The firm has offices in Sweden, Norway, Denmark, Poland, and the United Kingdom, in addition to Finland. Trade, industry, energy, and services are the core sectors in which the organization has a strong experience. Solteq's key skills include artificial intelligence, robotics, deep vision, service design, business intelligence and analytics, online services, eCommerce, and cloud services. Artificial intelligence and machine learning are all being integrated into the core of the company's products and services. Currently, around 650 IT specialists are employed at the company, and Solteq's revenue exceeded 69 million euros in 2021 (Solteq, 2022).

1.5 About the author

The author started his studies at JAMK University of Applied Sciences in August 2021 as a participant in the IT-Pro Double Degree Programme between JAMK and the University of Debrecen (Debreceni Egyetem) while also completing his degree at his home university.

In January 2022, the author began his practical training at Solteq as part of a 6-month Trainee program. The task of the author includes participating in the development and support work of the Omni commerce team while also being an active member of daily meetings and learning how to work and collaborate in an agile workflow. Learn the use of version control and other development tools.

During the training, the author learns how to use the broader range of tools of an HCL Commerce website, including order management, scheduled jobs management, and server maintenance works through several real-life tasks. Furthermore, he learns how to document the development process and how to communicate with the client regarding new features or fixes.

2 eCommerce

2.1 In general

The selling and purchase of products and services via the internet, as well as the online exchange of data and money for these transactions, is known as ecommerce (also known as electronic commerce or internet commerce) (Shopify, 2022). Ecommerce is frequently used to refer to the online selling of tangible products, although it actually refers to all online business activities. The development of eCommerce solutions has been made possible by the development of computer technology and technologies such as electronic payment, internet marketing, online transaction processing, electronic data exchange between businesses, and automated inventory management technologies. It is a subset of the wider electronic business (E-Business) sector, which covers all aspects of running a company online.

2.2 How it works

Customers may browse and purchase things or services from an online shop using their own devices that are linked to the servers through the internet.

When a customer places an order, their web browser starts a connection with the server that hosts the online store's website. The data from the order is given to the order manager, which then transmits it to databases which job is to track the levels of inventory, a merchant system that holds the information of payment details, and a bank server before returning it to the order manager. This is done to ensure that the business has enough merchandise and that the consumer has enough money to finish the purchase.

The order manager notifies the store's web server when the order has been confirmed, and the web server shows a message which purpose is to notify the customer about that the order has been processed and it will be shipped. To ensure that the product or service is delivered on time, the order manager will send order data to the warehouse or fulfillment center.

2.3 Impact of COVID-19 pandemic on eCommerce

Over the last two years, the global eCommerce sector has experienced a radical transformation. Many analysts agree that this shift in consumer behaviour has propelled the eCommerce industry forward by at least five years (Zidane, 2021). Consumers have become used to purchasing goods from the comfort of their own homes due in part to strict lockdowns and movement limitations. Furthermore, this change in buying habits appears to be long-term rather than temporary.

According to Statista’s research, eCommerce sales in Europe climbed by 20% in 2020, more than double the pre-pandemic gain from 2018 to 2019. By 2022, revenues are expected to approach $500,000 million (Figure 1). By 2025, revenue is estimated to expand at an annual rate of 11.35 percent, resulting in a market volume of US$5,726,101 million worldwide (Statista, 2021).

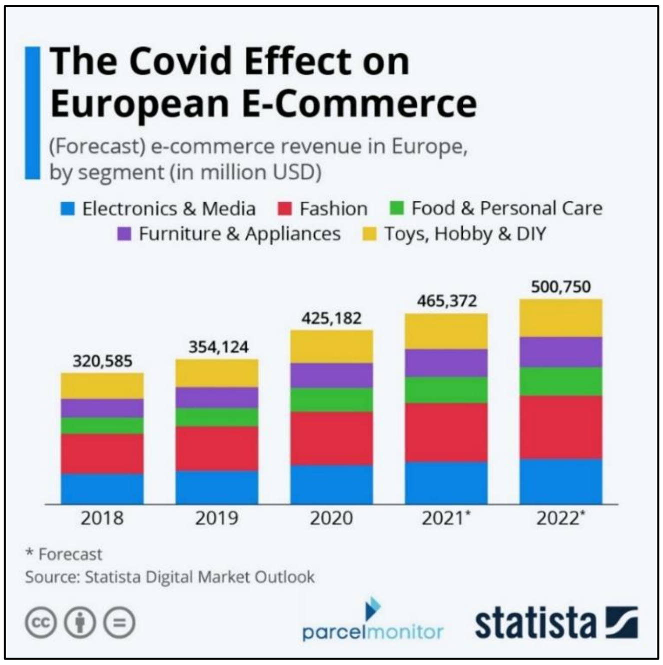


Figure 1. eCommerce revenue in Europe (Statista, 2021)

Observing all the historical and forecasted data, they show that the pandemic accelerated the annual growth of sales through electronic channels, which was caused not only by the restrictions and social distancing rules. However, as more consumers tried this easy way of shopping, they realized the convenience of purchasing through these channels so they got used to this way of shopping.

2.4 Business Models

An eCommerce business model is a method of an eCommerce organization is fundamentally built to reach clients and increase revenue. Ecommerce business models come in various shapes and sizes (Figure 2), allowing diverse types of enterprises to successfully position themselves in the market and reach out to their customers. In the next section, many types of business models are addressed in terms of who is the seller and the buyer counterpart throughout the transaction.

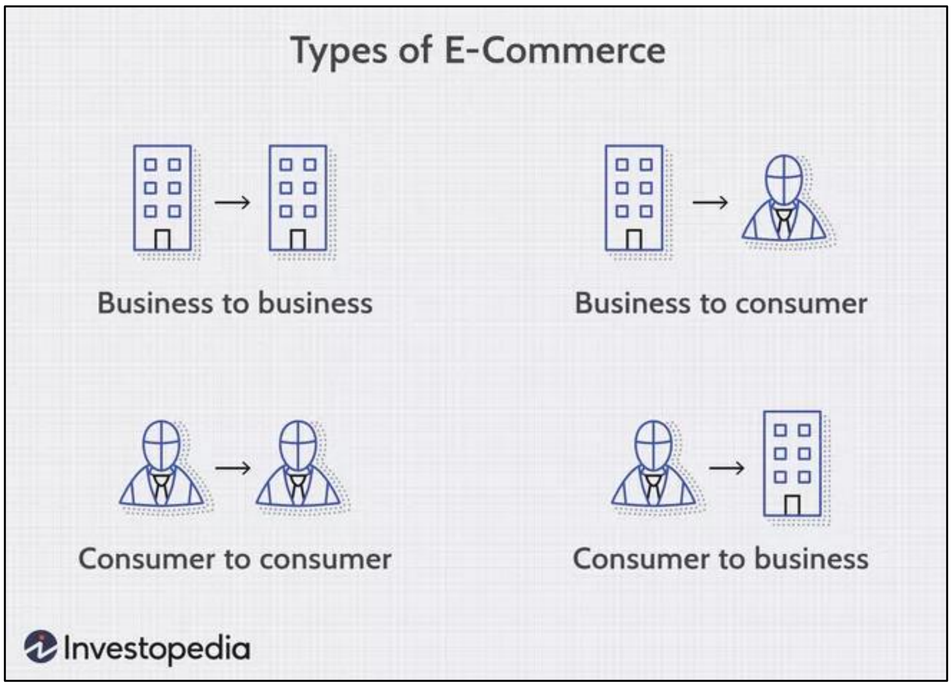


Figure 2. Types of E-Commerce (Investopedia, 2020)

2.4.1 Business to consumer

The B2C business model refers to a website that sells its items directly to customers. Customers can go over the products available on the website. When a consumer selects merchandise and places an order, the website sends an email notice to the business, which eventually ships the product to the client.

2.4.2 Business to Business

When a business is selling its goods to an intermediate buyer, who subsequently sells the product to another consumer, it is known as a B2B business model. For example, a wholesaler can put an order on another business own site, and then the goods are sold to a final client who visits one of the company's retail stores or websites.

2.4.3 Consumer to Consumer

Users may sell their assets, such as residential property, automobiles, motorcycles, or even rent a property. To do that they publish their information on a website that follows the C2C business model where they may or may not be charged for the service offered by the website that serves as a middleman between the parties. After seeing the advertisement on the website, another buyer may opt to purchase the first product in the search results. Therefore, there is a competition to be presented as first by the search engines.

2.4.4 Consumer to Business

During using a C2B business model a consumer checks out a webpage that displays various company listings that provide a specific service. The consumer can also determine how much approximately they want to spend for a particular service. Websites may be used to compare interest rates on personal loans or vehicle loans from different lenders. If a corporation can satisfy a customer's demands while staying under budget limits, it approaches the consumer and offers its services.

2.4.5 Business to Government

Businesses who are using the B2G business model provide products, services, or information to governments or agencies connected to a government. Through B2G networks, companies can offer different quotes for government projects or products that governments may purchase or use for their companies, making the process more competitive and lowering the price.

2.4.6 Government to Business

G2B's goal is to ease corporate difficulties by providing one-stop access to information and enabling digital communication. Furthermore, the government should make effective use of the data supplied and utilize commercial electronic transaction procedures.

2.4.7 Government to Citizen

To approach citizens in general, governments use G2C model websites. Auctions for automobiles, machinery and other items are supported by such websites. A website like this also offers services, including birth, marriage, and death certificate registration. The primary goal of G2C websites is to shorten the time it takes for citizens to get a wide variety of government services.

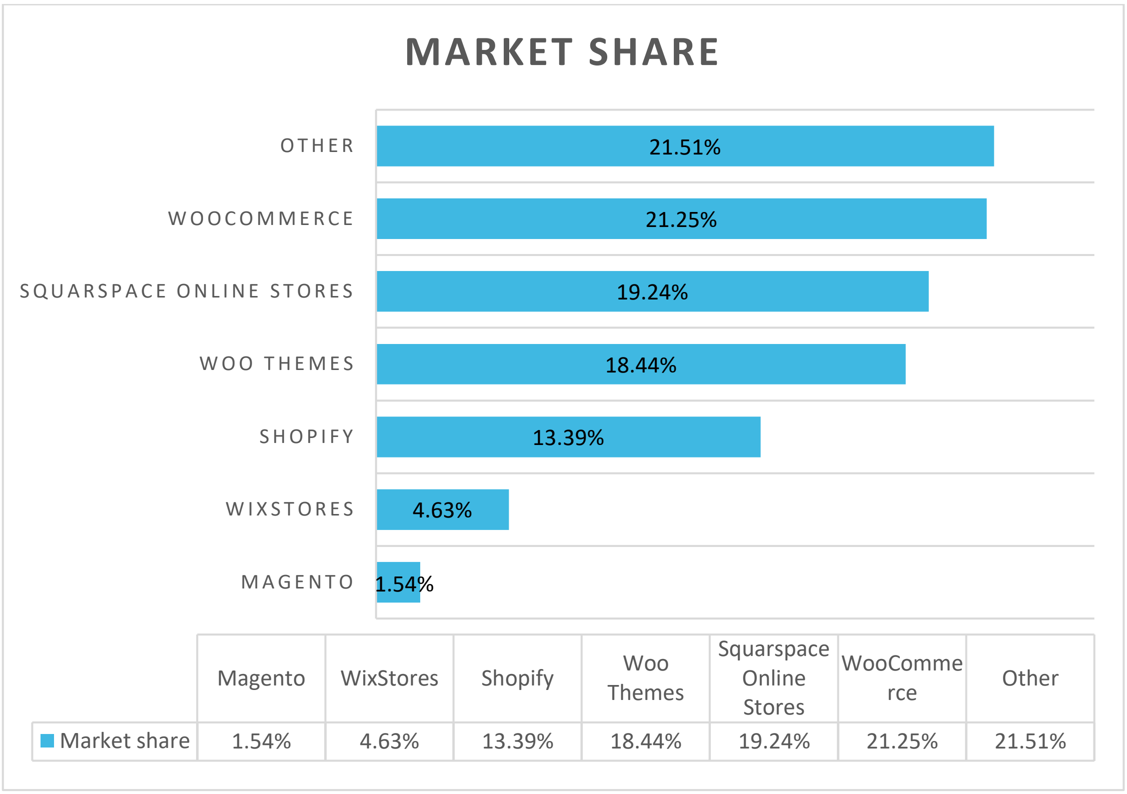
2.5 Platforms

Several platforms provide eCommerce solutions for customers with different needs and business models. Before building a commerce solution, one of the key steps is to compare the available platforms while comparing their features and prices regarding the owner's needs. These features can be the following: business model, marketing campaigns strategy, target audience, future vision, and price range.

The three fundamental types of eCommerce platforms are SaaS (software-as-a-service) platforms, PaaS (platform-as-a-service) platforms, and on-premises platforms. The internet is used by both SaaS and PaaS systems to provide eCommerce solutions. Software-only platforms are known as SaaS platforms. PaaS platforms are defined as software-as-a-service platforms with a hardware component (Adobe, 2021).

On-premise eCommerce systems are hosted locally by the merchant and managed by their IT staff, rather than just being set up by another supplier and accessible via the cloud. Smaller companies and those just getting started in eCommerce frequently look at SaaS and PaaS options. They offer professional setup and support, but they frequently demand a monthly fee for site access as well as transaction fees on each purchase. Businesses using on-premise solutions have more control over their eCommerce sites and may create unique storefronts (Adobe, 2021).

The market share of platforms providing these types of solutions can be seen in Figure 3. The data shows that Squarespace Online Stores and WooCommerce are the two leading ones, followed by Woo Themes and Shopify. The eCommerce business has risen considerably in recent years, resulting in higher adoption rates for various platforms. It is good to mention that WooCommerce can dominate the market thanks to it is being open-source and providing a highly flexible development environment.



	Magento	WixStores	Shopify	Woo Themes	Squarspace Online Stores	WooCommerce	Other
🟦 Market share	1.54%	4.63%	13.39%	18.44%	19.24%	21.25%	21.51%

Figure 3. Ecommerce Platforms' Market Share (Datanyze, 2022)

As Table 1. shows there are significant differences between the features of the most popular platforms as well as they have different approaches to pricing. Magento and HCL Commerce are among the more expensive choices, so they are widely used only by nationwide or multinational corporations with a more significant stream of revenue and wide product range. As it was mentioned before, WooCommerce is an open-source platform, so it is easy to find the appropriate solution from the extensions made by the community. Its use is also free to use and features over 1,000 plugins and hundreds more extensions. There are hundreds of premium add-ons available for different price rates. Many premium themes now include WooCommerce support as well as plugins that enable theme framework compatibility.

Table 1. Comparison between leading eCommerce platforms

	WooCommerce	**Shopify**	**WixStores**	**Magento**	**HCL Commerce**
Pricing	Free	Basic: $29/month, Shopify: $79/month, Advanced: $299/month	Free, Combo: $19/month, Unlimited: $25/month, Business Basic: $30/month	Magento Open Source: Free, Other versions: Custom pricing	Custom pricing
Flexibility	Customizable themes, Self-hosted, works with Woo Themes, can be used only with WordPress	Customizable themes, secure hosting and domain setup is included	Customizable themes on Wiz, contains basic ecom-merce tools	Highly customi-zable, self-hosted, best for big online stores	Customizable storefronts, cloud and self-hosting
Extensions	Massive library of WordPress ex-tensions	Huge library of extensions	Possible exten-sions, but limited com-pared to oth-ers	Massive library of extensions, recommended to be installed by developer	Widget extensions
Ease of use	Easy to use with minimal knowledge of CSS and HTML	Can be man-aged by non-technical staff	Drag and drop configuration	PHP developer or Magento consultant is needed	Java developer is needed
Customer Support	Limited support	24/7 support (phone, mail, live chat)	Support avail-able (phone, tickets)	Limited support	Support availa-ble(ticket), forum community

It is worth mention the solution of Shopify, which is a subscription-based platform that allows anybody to develop an online shop and sell their products. With the support of Shopify POS, a point-of-sale program, and accompanying hardware, Shopify store owners may also sell in physical locations. Merchants with online and physical operations may integrate their inventory and items to run multiple shops from a single account (Shopify, 2021).

2.6 Headless commerce

One of the newer trends in this market is called headless commerce. It is an eCommerce solution that stores, maintains, and delivers content without involving and concerning the frontend layer. A headless commerce platform decouples and removes the front end (or "head"), which is usually a template or a theme, leaving just the backend part. Developers may then utilize APIs to distribute things like items, blog entries, and customer reviews to any screen or device. At the same time, frontend developers can work on presenting the content material using whichever framework they choose. Basically, all the system's functional aspects can be handled programmatically. The production and maintenance of content components are included in this. The objective is to encapsulate business logic in cloud-deployed microservices. Then these microservices can be wrapped in whichever frontend user interface the user chooses. The customer could have different user interfaces for a retail store's point of sale, another for an eCommerce website, then another for a mobile app, and so on. Modern selling solutions are based on the notion of headless commerce (Honaman, 2021).

In Figure 4, the basic architecture of headless commerce can be observed. The backend layer has the connection with the databases (like Product Information Management – PIM data), while on the other side, there is an API layer between it and the frontend layer. With this architecture, the frontend development can be independent of the backend logic. Furthermore, different devices (like desktop, mobile, or even kiosk systems) can use the same backend system using API calls.

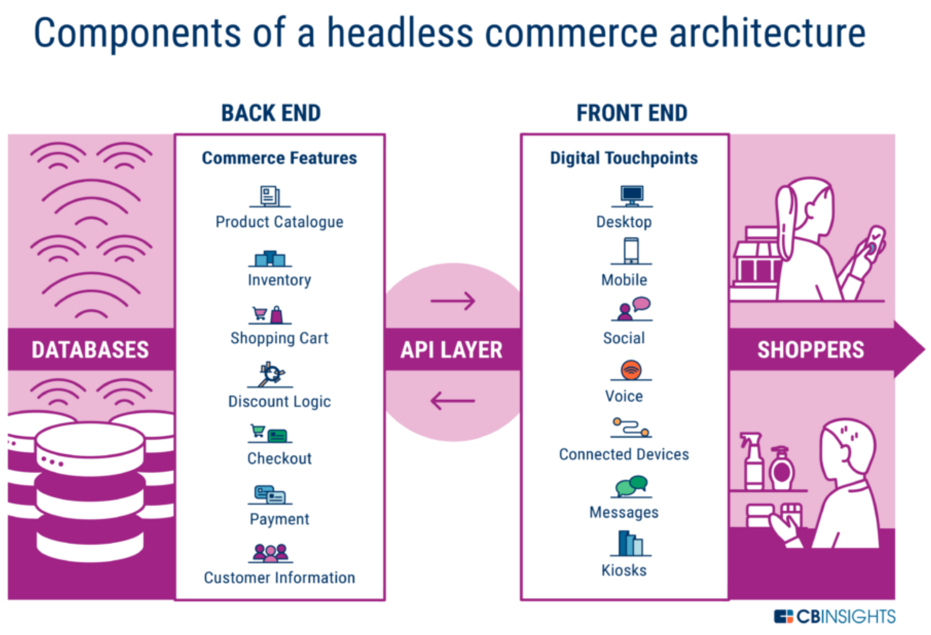


Figure 4. Headless commerce components (CB Insights, 2021)

Developing systems like this requires more sophisticated setups that frequently require the creation of unique code. To put it another way, standard functionalities must be rewritten to operate with the headless eCommerce system. Despite all of the advantages of an eCommerce system, it is not ready to use as a traditional system right away, but it requires quite a lot modification and custom features.

3 HCL Commerce

3.1 Introduction

HCL Commerce is one of the popular platforms of eCommerce, which is extensively used by Solteq. It includes flexible catalog management, multi-site implementation, order processing, and marketing capabilities.

It can be used to do business directly with consumers (B2C) and also with other businesses (B2B). It provides a unified management system for both types of stores lowering the total costs of ownership. In the past, it was a product of IBM (called WebSphere Commerce), and it was sold to HCL Technologies in July 2019. In December 2018, IBM and HCL Technologies signed an agreement under which HCL would purchase IBM’s WebSphere Commerce software line. The latest version (Version 9.1) was released in 2020. However, Version 8.0 was used for this project and called HCL WebSphere Commerce, referring to its former developer company.

3.2 Version 8 – HCL WebSphere Commerce

With IBM Web-Sphere Commerce V8.0 improvements, merchandisers and marketers may enhance consumer engagement, sales, and profitability by providing a better business user experience. It is improved to provide better customer service and assistance compared to earlier versions, while it incorporates prior WebSphere Commerce V7.0 feature pack capabilities and delivers a software platform update to the newest web application server, database, and integrated development environment. Version 8 has two released editions: Commerce Professional and Commerce Enterprise (IBM, 2022).

Version 8 was being used in the project’s environment at the time when this research and work was made.

3.2.1 WebSphere Commerce Professional

Midsized businesses may utilize this full business-to-consumer (B2C) cross-selling solution to create a tailored and consistent buying experience. The Professional edition includes capabilities such as precision marketing and merchandising, flexible business procedures, multivariate testing, search engine optimization, and personalization. These features may be used by businesses to attract, motivate, and understand their consumers at all stages of the customer experience. The Professional edition of WebSphere Commerce also supports the Extended Sites architecture, which was previously only available in WebSphere Commerce Enterprise (IBM, 2022).

3.2.2 WebSphere Commerce Enterprise

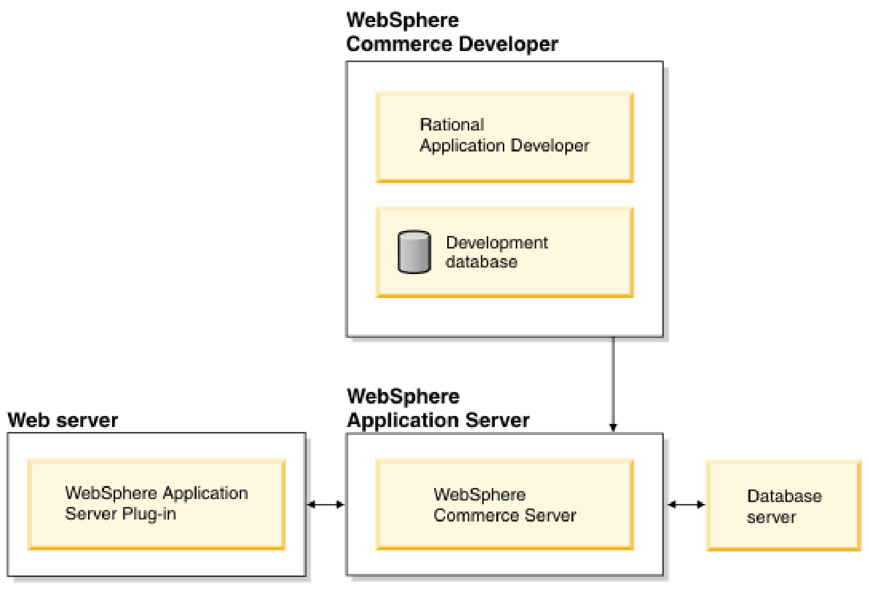
It is an omnichannel eCommerce platform that lets consumers buy from a business via online channels, on mobile, on social media, in stores, and over the phone. The Enterprise edition provides companies with merchandising tools, precision marketing, site search, customer experience management, catalog management and social commerce capabilities allowing them to engage consumers in a tailored and consistent way. The comprehensive B2C and B2B starter store has a responsive and optimized storefront for various platforms and formats, including web, mobile, and tablet (IBM, 2022).

3.3 Version 9.1 – HCL Commerce

Version 9.1 of the software enables a cloud-native architecture by using microservices and improved scalability options. It supports the continuous delivery (CD) model, while Continuous Delivery Update Packages help to install fixes and new functionalities easily. Migrating from Version 8 is also easy and is also supported from the code level. Version 9.1 improves on Version 9's containerized cloud-native architecture and adds support for Kubernetes deployment. The HCL Software Factory can also be used to create and preview a software solution before creating a basic Helm Chart, which is a solution designed to make managing Kubernetes applications easier. This version includes new logging and performance optimization apps, making it easier to manage and troubleshoot the webstore and its performance.

3.4 Common architecture

In Figure 5 the fundamental components of an HCL WebSphere Commerce (v8) site can be seen, showing the dataflow between the different software building blocks, including the web server, the WebSphere Application Server, and WebSphere Commerce Developer.



Source: HCL Product Documentation

Figure 5. Software components of HCL architecture (IBM, 2022)

The Web server is the initial point of interaction for inbound HTTP requests to an eCommerce application. It utilizes the WebSphere Application Server plugin to handle the connections between the two components in order to communicate efficiently with the WebSphere Application Server. HCL Commerce uses WebSphere Application Server (WAS) under the hood as it provides the connection between the users with the Java applications and servlets.

Because the WebSphere Commerce Server operates within the WebSphere Application Server, it can take advantage of many of the application server's functionalities. The database server is where most of the application's data, such as product and client information, is stored. Extensions to the application are often accomplished by altering or expanding the WebSphere Commerce Server base code. Furthermore, it may be required to store data in a database that is outside the scope of the WebSphere Commerce database model.

A development database is used in the WebSphere Commerce development environment. To perform database changes, developers can utilize their preferred database tools. Between the WebSphere Commerce instance and the WebSphere Commerce database Web-

Sphere Commerce offers a one-to-one mapping. It is not possible to run several WebSphere Commerce instances with the same database.

WebSphere Commerce is multichannel capable, which means it can handle transactions across several sales channels. Multiple presentation layers (which are responsible for showing views) are supported by the framework additions, which isolate control code from business logic.

WebSphere Commerce offers two channels: the web channel and the sales channel. The presentation for the web channel is composed of JSP pages, while the web controller layer utilizes Apache Struts. The Eclipse client technology is used for the sales channel. Eclipse views and editors are used to render the presentation layer. Controller calls use the business logic facade, a generic interface implemented as a stateless session bean, to invoke controller instructions regardless of the channel. To activate controller instructions independently of the channel, the business logic interface (a generic interface implemented as a stateless session bean) is used. WebSphere Commerce commands are used to implement the command layer. EJB 2.0 is supported by the persistence layer (IBM, 2022).

4 Order Management Systems

4.1 Introduction

A platform that tracks sales, orders, inventory, and fulfillment is known as an order management system. Order management allows the people, procedures, and partnerships required for products to reach their intended customers. It begins when a customer places an order and finishes when the goods are delivered to the buyer.

It allows a business to manage the whole fulfillment process, from order collection to inventory management to delivery visibility and service availability. The workflow of the process varies based on the business's needs, but a typical order management procedure has fundamentally three parts. The first is when the customer places the order (placement), and the company confirms it after checking the ordered items. The second is fulfillment when an employee from the warehouse verifies shipping information and packs the chosen items. During the third step (Inventory management step) the company keeps track of the inventory levels when they vary in response to company demands (IBM, 2022).

An Oder Management System can digitalize and automate all of the operations outlined above. It records all data and operations, including order entry, inventory management, fulfillment, and customer service after the sale. The visibility delivered by an OMS can benefit both the company and the buyer. Customers may check what time their order should arrive, and organizations can keep track of inventories in almost real-time.

4.2 Distributed Order Management

Distributed Order Management is a more complex part of the OMS that distributes order fulfillment to the most optimal place. It improves the OMS process' second stage (fulfillment). After examining each selling location's inventory (like each shop or warehouse what items has on hand), optimization happens through this management system.

This strategy is used to improve fulfillment so that purchases arrive on time and at the lowest possible cost to the client. Order splitting, order routing, shipping, inventory forecasting, reordering, and inventory management are all automated by distributed order management systems, which assist in coordinating the process and optimizing the whole supply chain (Skubana, 2019).

In Figure 6, the dataflow and the route of the messages can be observed between the storefront, the eCommerce system (marked as HCL Commerce in figure) and the Distributed Order Management System (OMS). After placing an order, the details of the items, the customer and other necessary information are transferred to the OMS system, which processes the order while also updating stock information for the selected items.

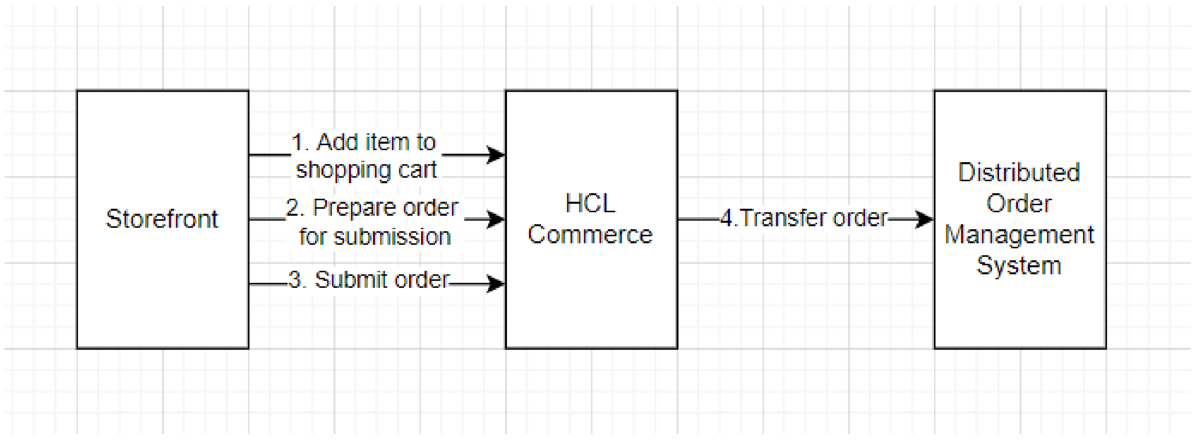

Storefront
1. Add item to shopping cart
2. Prepare order for submission
3. Submit order
HCL Commerce
4.Transfer order
Distributed Order Management System

Figure 6. Communication while placing an order in HCL commerce

Integrating a DOM system can mean several advantages for an eCommerce IT project. For example, as orders are maintained in a single system, whether the product is bought online, in-store, or via call centers customers have more control over their purchases, whether they are purchased online, in-store, or through customer service. Order centralization improves fulfillment efficiency by expanding fulfillment alternatives and increasing order and supplier visibility. Because of the increased visibility, DOM systems may prioritize supply requests based on particular client requirements. To suit changing client demands, purchases may be completed via any channel. Furthermore, improved monitoring capabilities allow for the creation of reports that are tailored to both business and consumer requirements. Defining the essential criteria allows for modifications and improvements to the existing fulfillment strategy, assuring future efficiency in completing orders successfully (IBM, 2022).

4.3 Dataflow

When the item’s Product Detail Page (PDP) is created, the current stock information is needed to show the item’s availability to the customer.

Inside the test and production environment of the project, when inventory data is needed it is first checked from the Dynamic Cache, which can store the inventory data for frequently visited items. This means that it is not required to check the OMS, which accelerates the speed of obtaining inventory information. When data is requested by the Dynamic cache, the system checks if it exists in the Cache database. If the required information is not present neither there, a call is transmitted to the OMS, which responds with the required data. This is stored in the cache database and also transmitted to the UI, which now can display the current stock information regarding the item selected by the user. During this process, the messages are in the form of XML files. The visual representation of this dataflow can be seen in Figure 7.

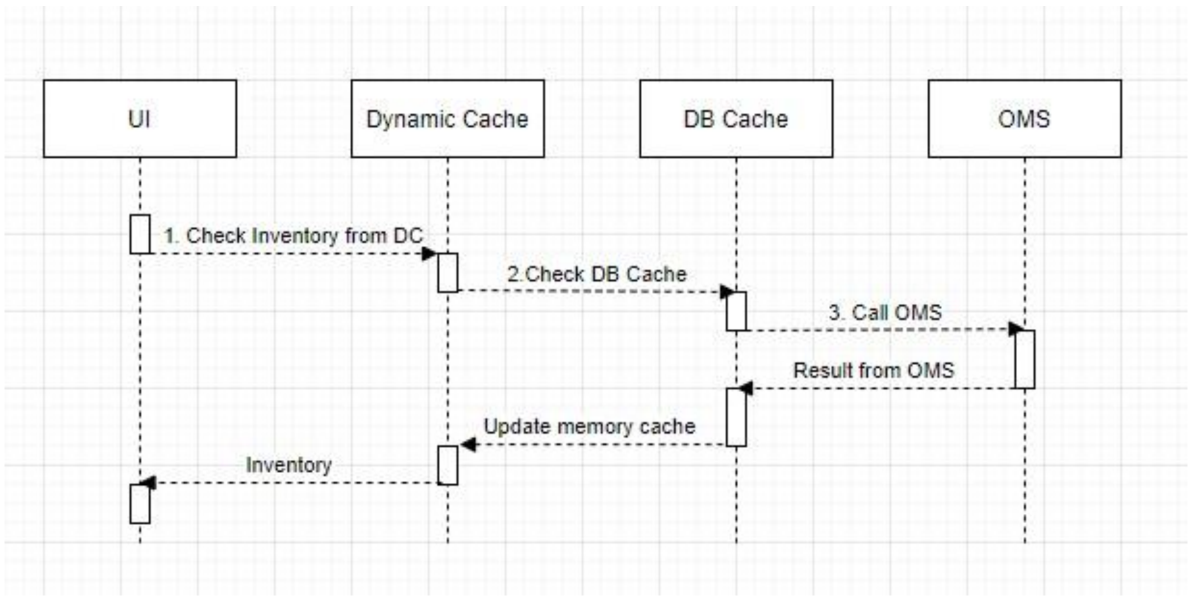


Figure 7. OMS dataflow inside the production environment

All in all, the data is acquired from the OMS, which is not present in the development environment. This makes the testing of several features hard, for example the process of updating the inventory and even placing an order with simplified inventory data. In Figure 7 the Dynamic Cache, the DB Cache, and OMS can be regarded as one unified system, which allows real-time data regarding stock information to be displayed on the website.

4.4 OMS Simulator

The external OMS simulator acts as a placeholder for a fully complete DOM system, with generic inventory levels created and integrated into the storefront stock availability features. For this project it was provided by IBM and it was written using Java language. After HCL purchased the WebSphere commerce software from IBM, there was a compatibility issue between the site that uses HCL Commerce and the OMS Simulator provided by IBM.

As described in Section 4.3, the test and production environment have their own dedicated OM system, while the SOAP messages are transported between the site and OM (including cached data), but this process is not implemented in the development environment of this particular Solteq project, making it quite challenging to test features regarding stock and items' availability.

In Figure 8 the flow of messages between the site's UI and OMS Simulator can be seen. For example, when a product detail page is opened, the UI sends a message to the OMS Simulator about checking the inventory details for a specific product. It processes that request and sends back the response to the UI containing inventory details.

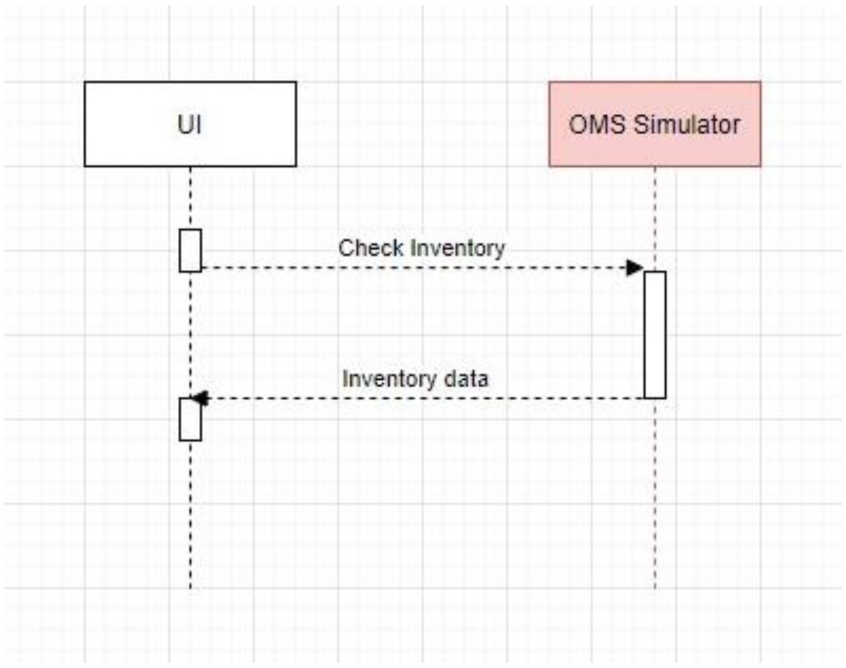


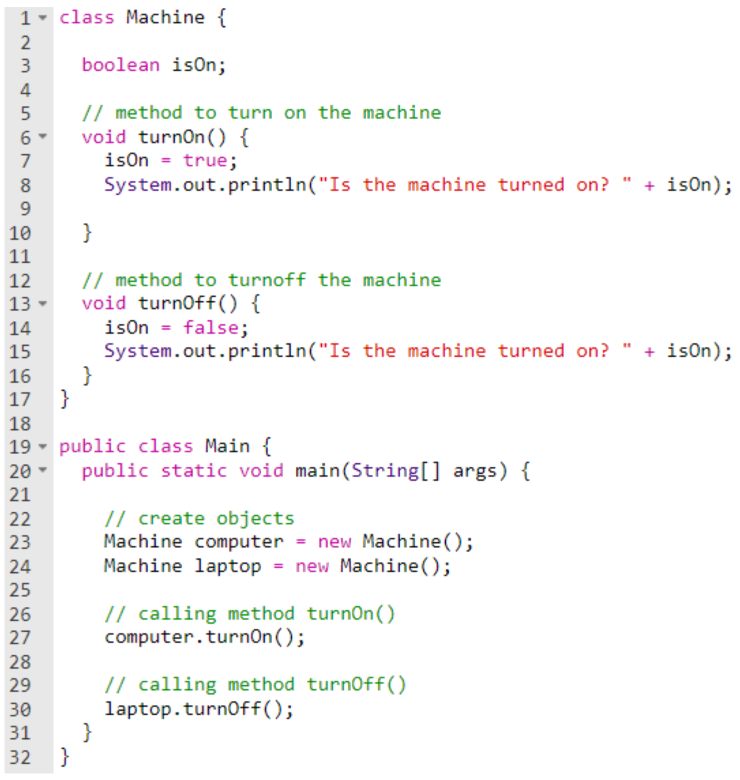
Figure 8. Dataflow between UI and OMS Simulator

5 Technical background

5.1 Java

Java has long been one of the most widely used programming languages. It is not considered a true object-oriented since it allows basic data types, despite the fact that it is object-oriented. The programs are first compiled into byte code (machine-independent code) and then run on the Java Virtual Machine (JVM), which is unaffected by the underlying architecture.

Although the syntax of Java (Figure 9) is similar to that of C/C++, it lacks low-level programming capabilities like pointers. Java applications are also always specified in terms of classes and objects. It is used in a variety of applications, including mobile apps (Android is built on Java as well), desktop apps, web apps, client-server apps, commercial software, and more. In this project it is used in the development of the HCL commerce site, and the OMS Simulator is also based on Java language.



```java
class Machine {

  boolean isOn;

  // method to turn on the machine
  void turnOn() {
    isOn = true;
    System.out.println("Is the machine turned on? " + isOn);

  }

  // method to turnoff the machine
  void turnOff() {
    isOn = false;
    System.out.println("Is the machine turned on? " + isOn);
  }
}

public class Main {
  public static void main(String[] args) {

    // create objects
    Machine computer = new Machine();
    Machine laptop = new Machine();

    // calling method turnOn()
    computer.turnOn();

    // calling method turnOff()
    laptop.turnOff();
  }
}
```

Figure 9. Sample of a Java code with classes

5.2 XML

5.2.1 Introduction

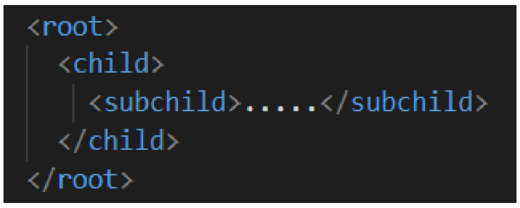
During this project, XML-based files are widely used to communicate and transport data between the OMS and the HCL commerce site and acquire the allowed fields of the SOAP requests for the OMS Simulator described in WSDL files.

The Extensible Markup Language (XML) is a text-based format for encoding structured data, such as documents, data, configuration, transactions, and invoicing. It was created from an older standardized format known as SGML in order to make it more Web-friendly (W3C, 2008).

XML offers a broad spectrum of applications: it is now used by several apps and devices to manage, organize, store, transfer and display data. It is commonly used in both B2B and B2C data transfers. Office file formats, such as Microsoft Office and Google Docs, are likewise based on XML. As XML maintains data in plain text format, it is unaffected by the platform which is used to open it, and it can be exported, imported, or even relocated considerably more quickly.

5.2.2 Structure

Element trees are utilized to build XML documents. An XML tree starts with a root element and then branches out to child elements as it can be seen in Figure 10. They may all have subcomponents or child elements. The labels parent, child, and sibling are used to indicate component relationships. In an XML document, there can be only one root element.



```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

Figure 10. Structure of an XML file

Starting from the root, all subsequent branches and sub-branches can be visited by applying tree structure. The parsing process begins at the root, then continues down the first branch to an element, then down the second branch and so on to the leaf nodes.

5.3 SOAP messages

5.3.1 In general

Microsoft designed the Simple Objects Access Protocol (SOAP) in 1998 as an online communication protocol. Today it is often used to offer web services and transfer data via HTTP or HTTPS protocols. It is not limited to them, however. Unlike the REST style, SOAP only accepts XML data and follows predetermined standards such as message structure, encoding requirements and a process request and response pattern.

5.3.2 Structure

A traditional SOAP document includes a SOAP envelope, a SOAP header (which is optional) and a SOAP body. As mentioned, the header part is optional and it carries information regarding routing, which allows it to be transferred through the intermediate node before it arrives at the final destination (Oracle, 2022).

The root element of the XML document that represents the message is the envelope (Figure 11). It establishes the structure for how and by whom the communication should be handled. The SOAP processor recognizes that the XML is a SOAP message once it finds the Envelope element and may then search for the message’s subcomponents.

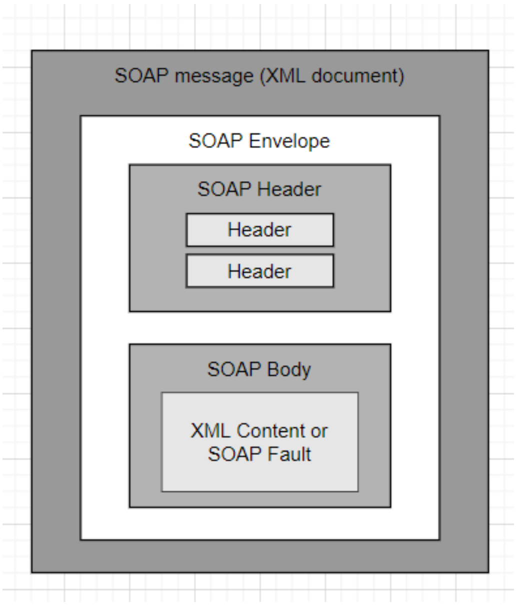


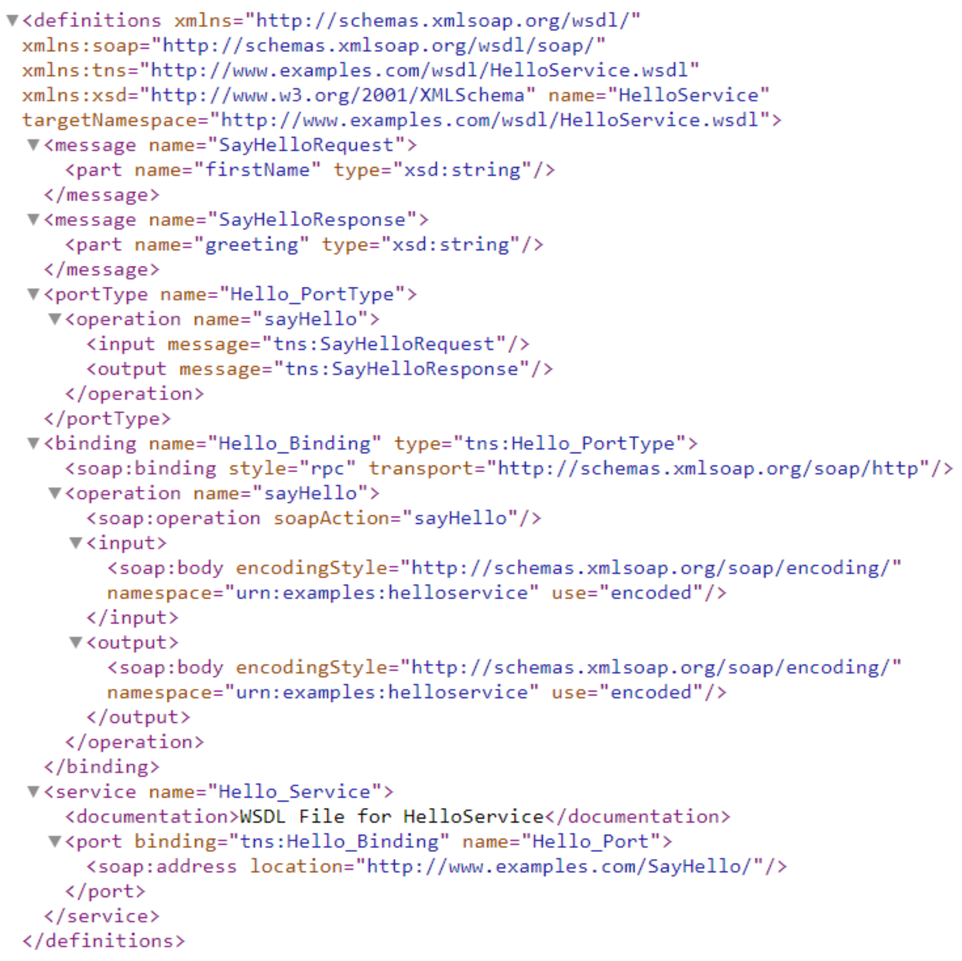
Figure 11. Structure of SOAP message

A SOAP message's header is a basic approach for adding features. It can have any number of child components, each of which defines a protocol extension. The header child elements can specify authentication, transaction, and localization information, among other things. Without a previous agreement, the software that handles the message may utilize this approach to designate who should interact with a component and whether it is required or optional. The message's body is a container containing necessary information meant for the message's final receiver.

5.4 WSDL

Web Service Description Language (WSDL) is an XML-based describing language for web services. It is used to define the capabilities of a SOAP-based web service. It provides a simple way for service providers to declare the basic syntax of requests to their systems, independent of the runtime technology utilized. WSDL files are frequently used while testing SOAP-based services. SoapUI uses WSDL files to produce test requests, assertions, and mock services. WSDL files provide a variety of SOAP message features, such as whether elements and attributes are required or optional, whether an element or property may appear many times and, if necessary, the specific arrangements and sequences of components (SoapUI, 2022). The structure and composition of an example WSDL file are shown in Figure 12.

Compared to REST, SOAP employs WSDL for consumer-provider communication, whereas REST uses only XML or JSON to deliver and receive data. WSDL is a static document that specifies a connection information between a client and a service.

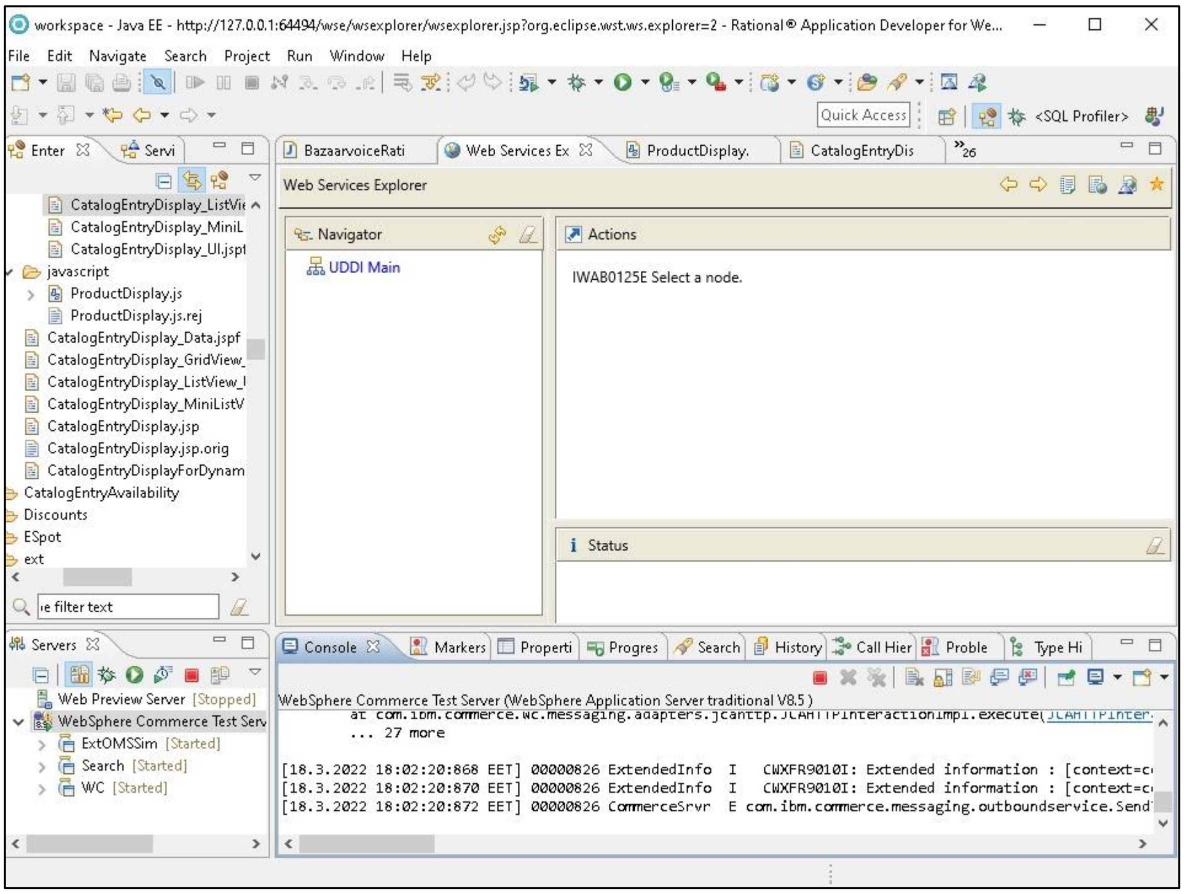


```
▼<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="HelloService"
 targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl">
  ▼<message name="SayHelloRequest">
     <part name="firstName" type="xsd:string"/>
   </message>
  ▼<message name="SayHelloResponse">
     <part name="greeting" type="xsd:string"/>
   </message>
  ▼<portType name="Hello_PortType">
    ▼<operation name="sayHello">
       <input message="tns:SayHelloRequest"/>
       <output message="tns:SayHelloResponse"/>
     </operation>
   </portType>
  ▼<binding name="Hello_Binding" type="tns:Hello_PortType">
     <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    ▼<operation name="sayHello">
       <soap:operation soapAction="sayHello"/>
      ▼<input>
         <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         namespace="urn:examples:helloservice" use="encoded"/>
       </input>
      ▼<output>
         <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         namespace="urn:examples:helloservice" use="encoded"/>
       </output>
     </operation>
   </binding>
  ▼<service name="Hello_Service">
     <documentation>WSDL File for HelloService</documentation>
    ▼<port binding="tns:Hello_Binding" name="Hello_Port">
       <soap:address location="http://www.examples.com/SayHello/"/>
     </port>
   </service>
 </definitions>
```

Figure 12. WSDL example file

5.5 Integrated Development Environment

During development, an Eclipse-based IBM Rational Application Developer (RAD) for Web-Sphere was used as the Integrated Development Environment with Runtime Analysis Tools (RAT) which helped to analyze the performance and find the possible bottleneck in the project of the client's eCommerce store. The IDE includes visual design, construction, testing, analysis and deployment capabilities for a variety of applications, including Java, Java EE and web/REST services. The user interface of IBM Rational Application Developer can be seen in Figure 13.


workspace - Java EE - http://127.0.0.1:64494/wse/wsexplorer/wsexplorer.jsp?org.eclipse.wst.ws.explorer=2 - Rational® Application Developer for We...
File Edit Navigate Search Project Run Window Help
Quick Access
<SQL Profiler>
Enter
Servi
CatalogEntryDisplay_ListVi
CatalogEntryDisplay_MiniL
CatalogEntryDisplay_UI.jspf
javascript
ProductDisplay.js
ProductDisplay.js.rej
CatalogEntryDisplay_Data.jspf
CatalogEntryDisplay_GridView_
CatalogEntryDisplay_ListView_I
CatalogEntryDisplay_MiniListV
CatalogEntryDisplay.jsp
CatalogEntryDisplay.jsp.orig
CatalogEntryDisplayForDynam
CatalogEntryAvailability
Discounts
ESpot
ext
ie filter text
BazaarvoiceRati
Web Services Ex
ProductDisplay.
CatalogEntryDis
»26
Web Services Explorer
Navigator
UDDI Main
Actions
Status
Servers
Web Preview Server [Stopped]
WebSphere Commerce Test Serv
ExtOMSSim [Started]
Search [Started]
WC [Started]
Console
Markers
Properti
Progres
Search
History
Call Hier
Proble
Type Hi

IWAB0125E Select a node.

WebSphere Commerce Test Server (WebSphere Application Server traditional V8.5)

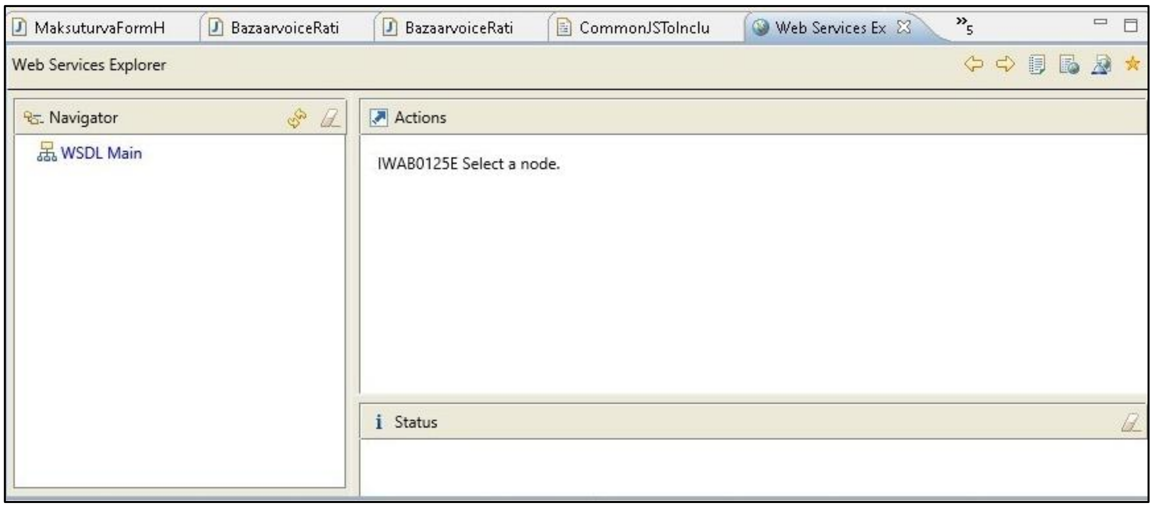
```
        at com.ibm.commerce.wc.messaging.adapters.jcahttp.JCAHTTPInteractionImpl.execute(JCAHTTPInter
        ... 27 more

[18.3.2022 18:02:20:868 EET] 00000826 ExtendedInfo  I   CWXFR9010I: Extended information : [context=c
[18.3.2022 18:02:20:870 EET] 00000826 ExtendedInfo  I   CWXFR9010I: Extended information : [context=c
[18.3.2022 18:02:20:872 EET] 00000826 CommerceSrvr  E com.ibm.commerce.messaging.outboundservice.Send
```

Figure 13. IBM Rational Application Developer

5.5.1 Loading WSDL in IDE

Eclipse has a built-in tool called Web Service Explorer, which is a JSP Web application that runs on Eclipse's Apache Tomcat servlet engine. It is integrated into Eclipse on two levels: visually as it runs in the embedded browser and logically because it is a thread in the Eclipse JRE. To open in Eclipse: Click Run > Launch the Web Services Explorer.

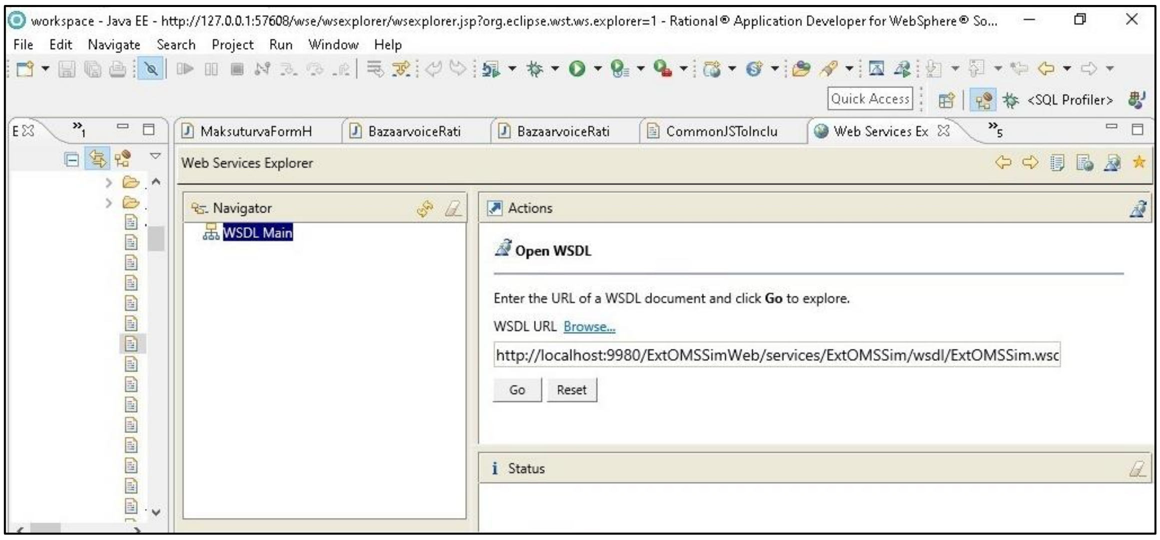
After opening it click the **WSDL Page** icon in the top right of the Web Service Explorer page, after this a window like in Figure 14 should appear.



IWAB0125E Select a node.

Figure 14. Web Service Explorer inside IDE

After this, click WSDL Main on the left panel. This displays Open WSDL in the Actions pane of the Web Service Explorer, where the URL to the WSDL file can be copied into the WSDL URL field as it can be seen in Figure 15.



workspace - Java EE - http://127.0.0.1:57608/wse/wsexplorer/wsexplorer.jsp?org.eclipse.wst.ws.explorer=1 - Rational® Application Developer for WebSphere® So...

Open WSDL

Enter the URL of a WSDL document and click **Go** to explore.

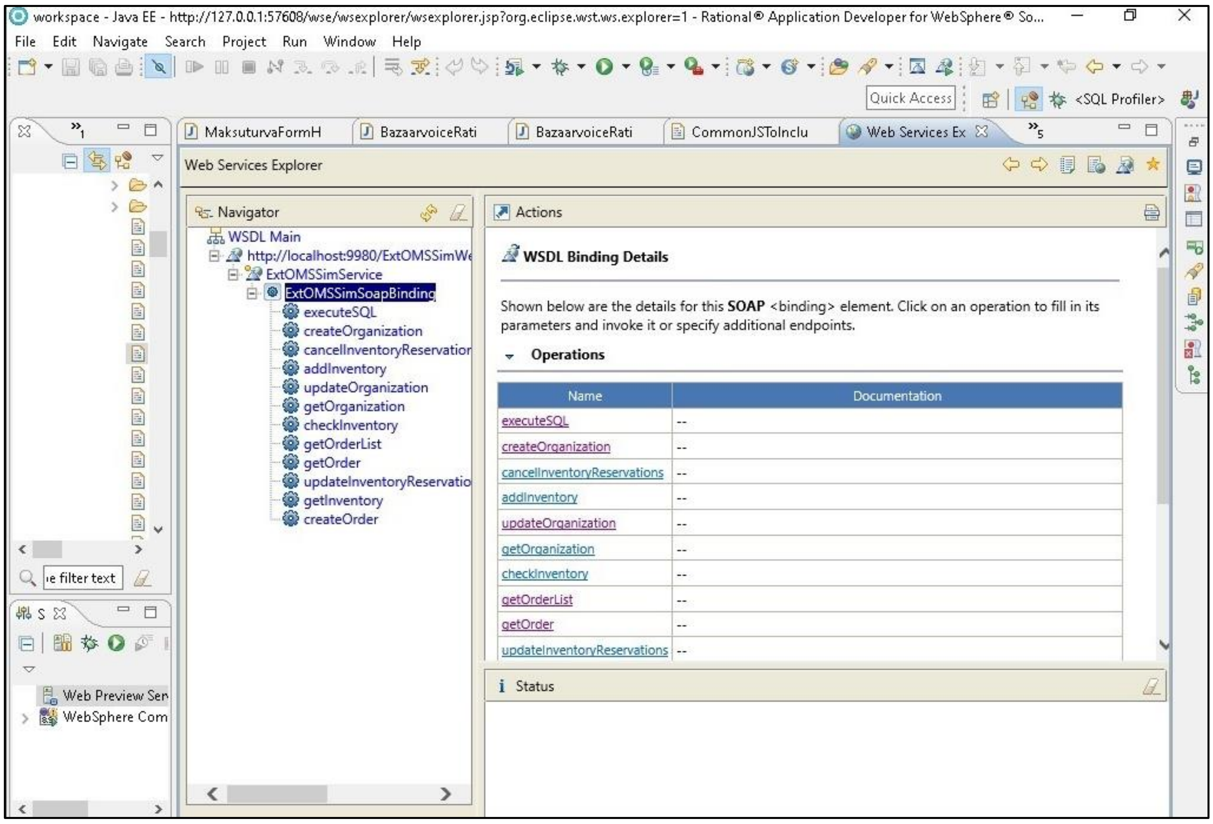
WSDL URL Browse...

http://localhost:9980/ExtOMSSimWeb/services/ExtOMSSim/wsdl/ExtOMSSim.wsc

Go Reset

Figure 15. Opening WSDL URL in IDE

After clicking the Go button, the next window can be seen as it is shown in Figure 16.



WSDL Binding Details

Shown below are the details for this **SOAP** <binding> element. Click on an operation to fill in its parameters and invoke it or specify additional endpoints.

Operations

Name	Documentation
executeSQL	--
createOrganization	--
cancelInventoryReservations	--
addInventory	--
updateOrganization	--
getOrganization	--
checkInventory	--
getOrderList	--
getOrder	--
updateInventoryReservations	--

Figure 16. The result after opening WSDL URL in IDE

Here the different SOAP requests are loaded from the WSDL file and can be used to send requests to the server, including different values to the changeable fields. The response is also shown under the Status section if it is received.

6 Solutions

6.1 Setting up the Simulator

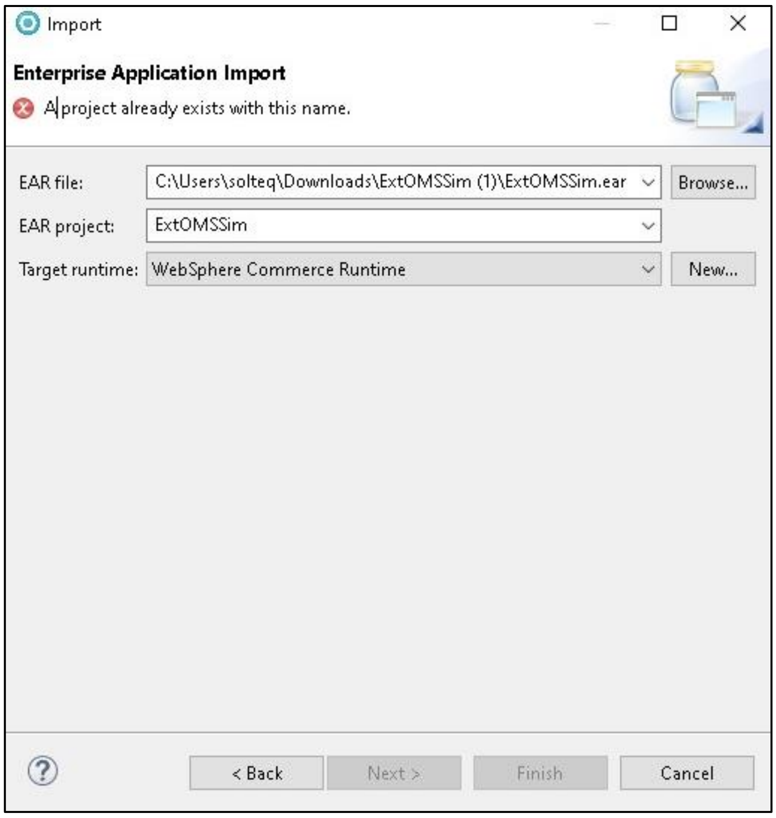
6.1.1 In general

One type of the found solution is to simulate an OM inventory with randomized items and quantities, including the name and part numbers.

During this process, the simulator provided by IBM is used as it is described in the official documentation of HCL WebSphere Commerce, which was introduced in Section 4.4. Before observing how it works, it is needed to be installed in an environment that already contains the servers which are needed to run an HCL commerce site. The installation should take place in a development environment as the simulator does not provide actual data, it is only for testing purposes.

The first step is downloading the IBM-provided file from their documentation site. The file's name is ExtOMSSim.ear, which is a compressed enterprise archive (EAR) file containing the libraries, enterprise beans, and JAR files that the application requires for deployment.

After this step, it was imported into RAD workspace: File > Import > Java EE > EAR file
Here the file path to the OMS Sim was selected, as can be seen in Figure 17.


Import

Enterprise Application Import

A project already exists with this name.

EAR file: C:\Users\solteq\Downloads\ExtOMSSim (1)\ExtOMSSim.ear

EAR project: ExtOMSSim

Target runtime: WebSphere Commerce Runtime

Figure 17. Importing an EAR file in IDE

The next step was to set the properties inside the WebSphere Application Server Administrative Console. For this, navigating to Servers > Application servers > server1 was done. After expanding the Web Container Settings and selecting Web container transport chains, a new transport chain was created with the properties shown in Table 2.

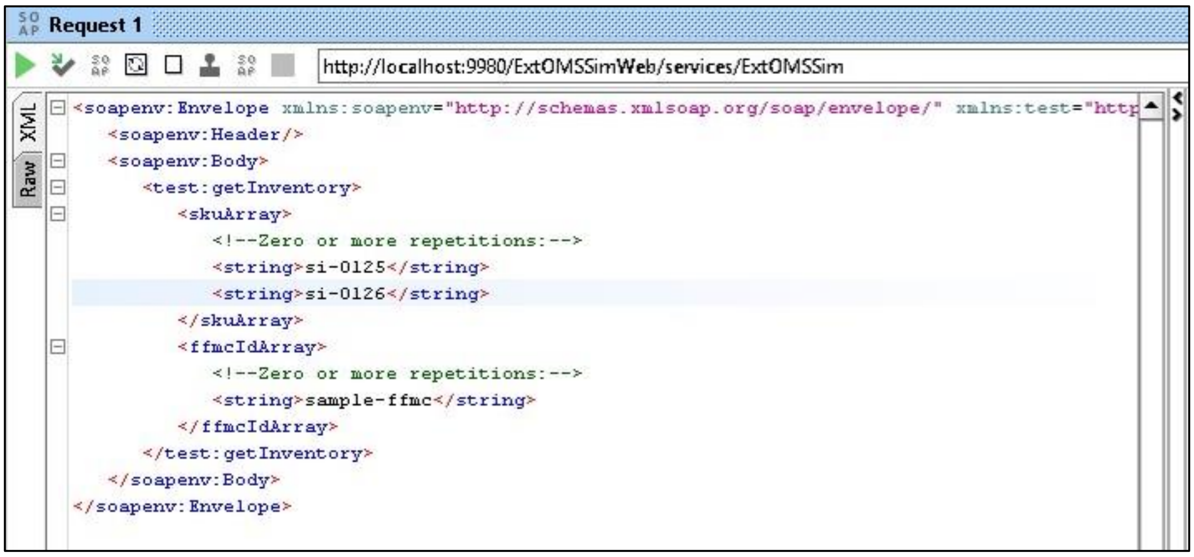
Table 2. Transport chain properties for OMS Simulator setup

Property	**Value**
Transport chain name	ExtOMSSimWeb
Port name	ExtOMSSimWeb
Host	*
Port	9980

Then the server should was and the external OMS Simulator was successfully enabled. To ensure that the set up process was done successfully, the following URL was used to access the WSDL file provided by the running simulator:
http://localhost:9980/ExtOMSSimWeb/services/ExtOMSSim/wsdl/ExtOMSSim.wsdl

6.1.2 Messages of the simulator

As mentioned earlier, the external simulator uses Soap messages for communication. First, it listens on port 9980 for requests from any source on the localhost. One of the messages is about sending a request for getting information about the inventory of a selected product (getInventory request from WSDL file). Its structure is shown in Figure 18.


Request 1

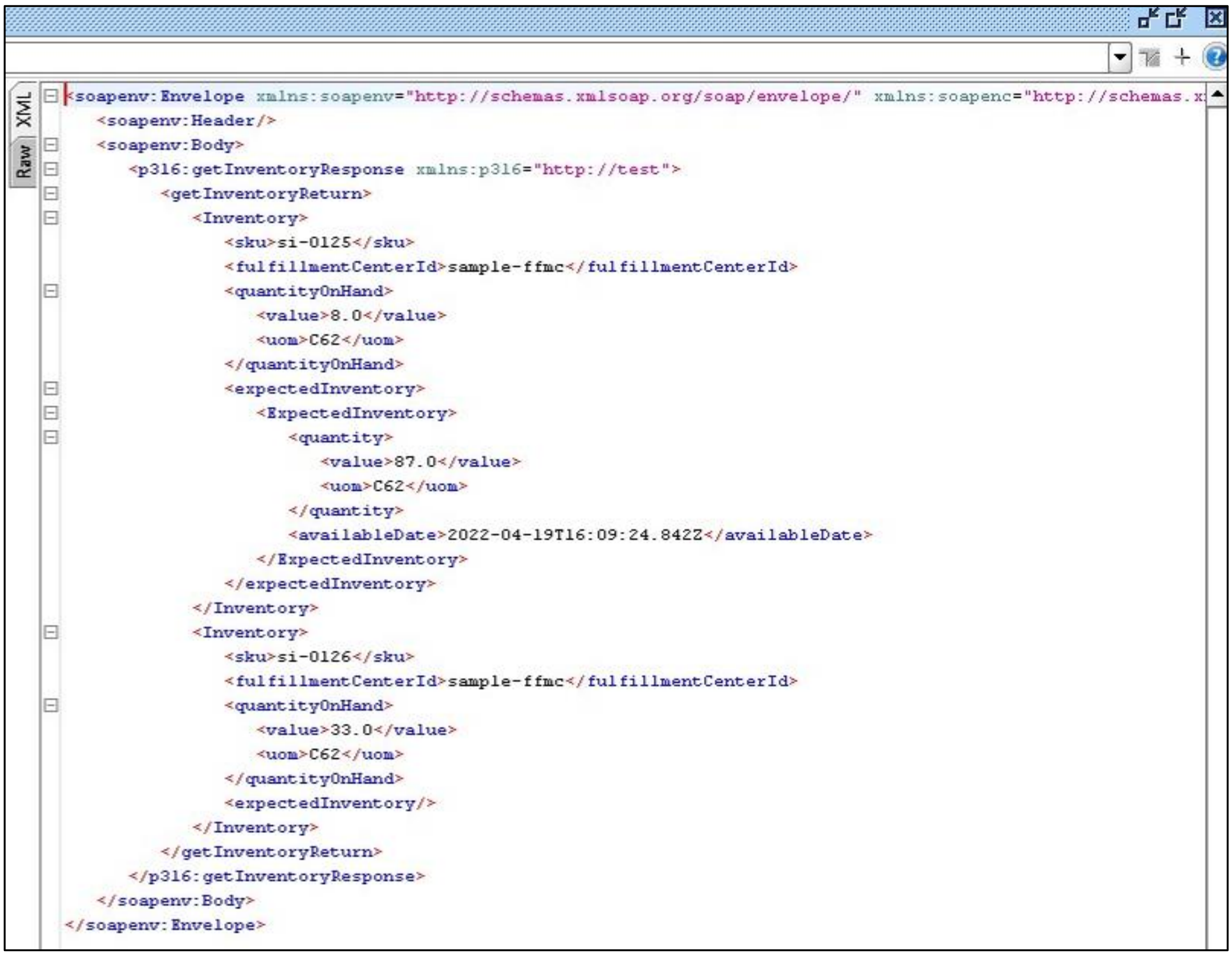
http://localhost:9980/ExtOMSSimWeb/services/ExtOMSSim

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:test="http
   <soapenv:Header/>
   <soapenv:Body>
      <test:getInventory>
         <skuArray>
            <!--Zero or more repetitions:-->
            <string>si-0125</string>
            <string>si-0126</string>
         </skuArray>
         <ffmcIdArray>
            <!--Zero or more repetitions:-->
            <string>sample-ffmc</string>
         </ffmcIdArray>
      </test:getInventory>
   </soapenv:Body>
</soapenv:Envelope>
```

Figure 18. Request sent to OMS Simulator

Firstly, an array of Stock Keeping Unit (SKU) numbers should be specified, which in the example above contains only two strings/items (si-0125, si-0126). After that, an array of fulfillment center Ids should be specified (ffmcIdArray), on the example above it is specified as ”sample-ffmc”.

After the request is sent, the following reply message arrives from the OMS Simulator, shown in Figure 19. The inventory data (like the quantity inside the fulfillment center and unit of measurement) is generated randomly according to the used strings, so these returned values can change from request to request in case of the input strings are changed. When the request is received by the OMS Simulator it processes the input fields and generates the output values based on the input values. The return value for the date of availability changes according to the given date and time used in the request.



```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:soapenc="http://schemas.x
   <soapenv:Header/>
   <soapenv:Body>
      <p316:getInventoryResponse xmlns:p316="http://test">
         <getInventoryReturn>
            <Inventory>
               <sku>si-0125</sku>
               <fulfillmentCenterId>sample-ffmc</fulfillmentCenterId>
               <quantityOnHand>
                  <value>8.0</value>
                  <uom>C62</uom>
               </quantityOnHand>
               <expectedInventory>
                  <ExpectedInventory>
                     <quantity>
                        <value>87.0</value>
                        <uom>C62</uom>
                     </quantity>
                     <availableDate>2022-04-19T16:09:24.842Z</availableDate>
                  </ExpectedInventory>
               </expectedInventory>
            </Inventory>
            <Inventory>
               <sku>si-0126</sku>
               <fulfillmentCenterId>sample-ffmc</fulfillmentCenterId>
               <quantityOnHand>
                  <value>33.0</value>
                  <uom>C62</uom>
               </quantityOnHand>
               <expectedInventory/>
            </Inventory>
         </getInventoryReturn>
      </p316:getInventoryResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

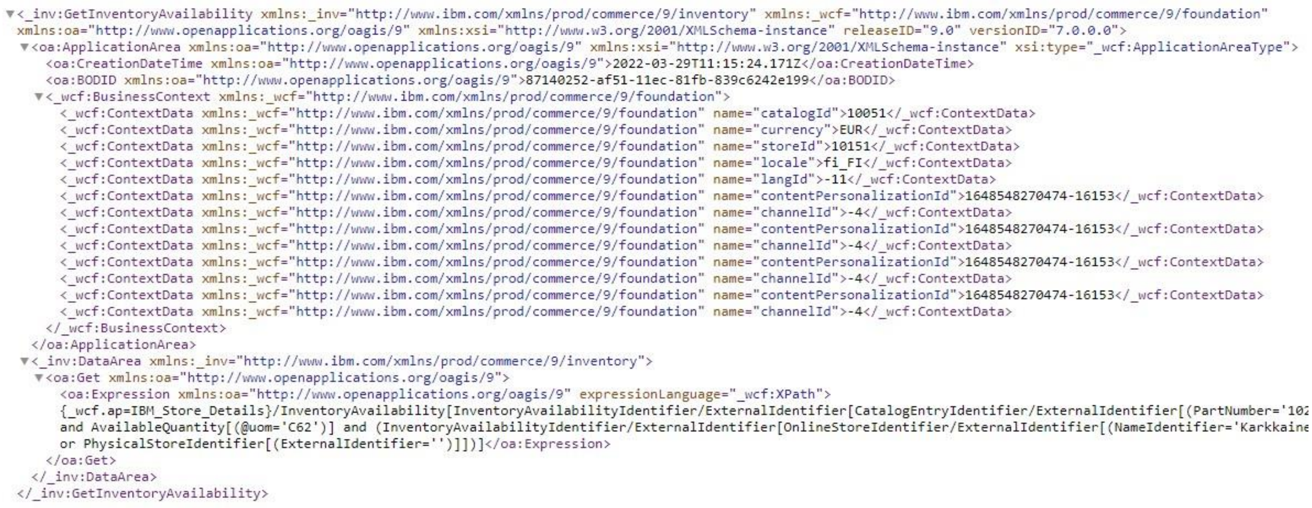
Figure 19. Response sent by OMS Simulator

As mentioned above, the returned data is different for every unique input string as the HCL site does not require fixed data, this is only needed for simulating a constantly changing inventory and order management system imitating the real world.

The information, which is encoded in SOAP responses (like in Figure 19) is meant to be digested and processed by the HCL storefront to show real-time data on the detail page of each item regarding the availability and quantity on each pick-up point, store or fulfillment center.

6.1.3 Messages of the OM System

After observing the SOAP messages of the OMS Simulator, the messages of the actual OM was looked at to compare the simulator messages to the real-life communication regarding getting inventory information for a specified item. The request sent by the site to OM can be seen in Figure 20.

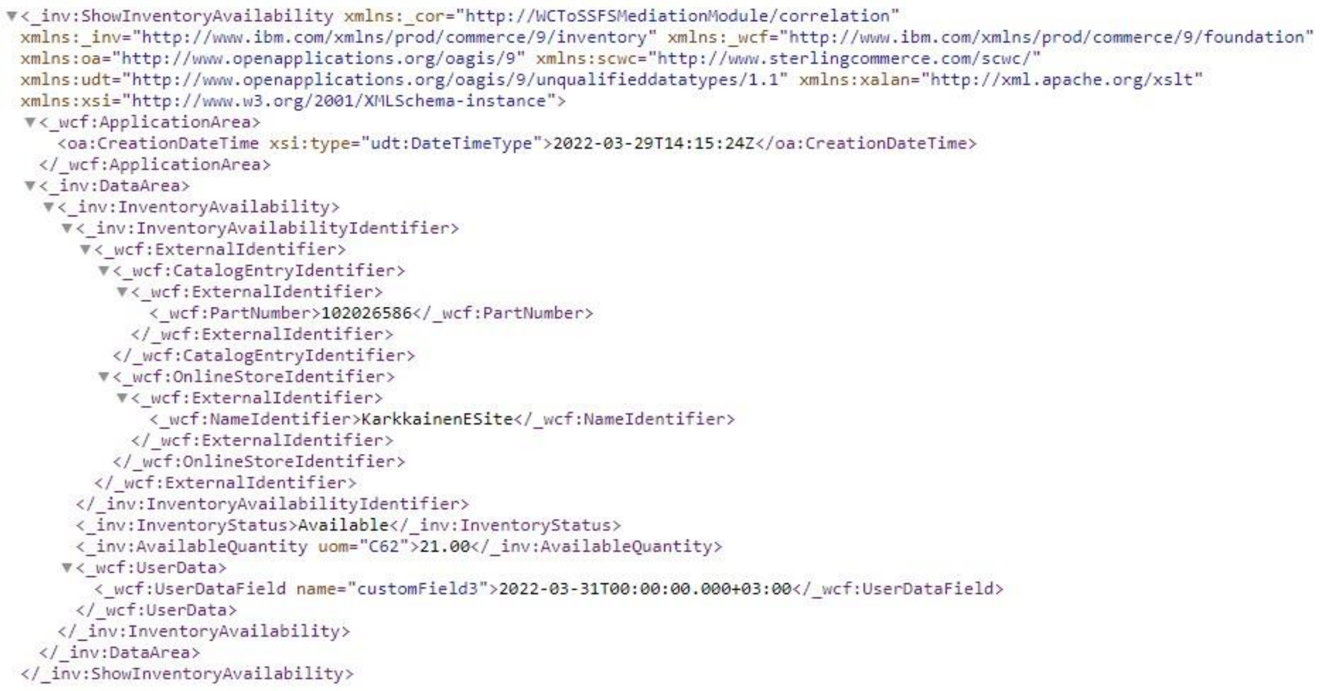


```
▼<_inv:GetInventoryAvailability xmlns:_inv="http://www.ibm.com/xmlns/prod/commerce/9/inventory" xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
 xmlns:oa="http://www.openapplications.org/oagis/9" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" releaseID="9.0" versionID="7.0.0.0">
  ▼<oa:ApplicationArea xmlns:oa="http://www.openapplications.org/oagis/9" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="_wcf:ApplicationAreaType">
     <oa:CreationDateTime xmlns:oa="http://www.openapplications.org/oagis/9">2022-03-29T11:15:24.171Z</oa:CreationDateTime>
     <oa:BODID xmlns:oa="http://www.openapplications.org/oagis/9">87140252-af51-11ec-81fb-839c6242e199</oa:BODID>
   ▼<_wcf:BusinessContext xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation">
      <_wcf:ContextData xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation" name="catalogId">10051</_wcf:ContextData>
      <_wcf:ContextData xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation" name="currency">EUR</_wcf:ContextData>
      <_wcf:ContextData xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation" name="storeId">10151</_wcf:ContextData>
      <_wcf:ContextData xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation" name="locale">fi_FI</_wcf:ContextData>
      <_wcf:ContextData xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation" name="langId">-11</_wcf:ContextData>
      <_wcf:ContextData xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation" name="contentPersonalizationId">1648548270474-16153</_wcf:ContextData>
      <_wcf:ContextData xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation" name="channelId">-4</_wcf:ContextData>
      <_wcf:ContextData xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation" name="contentPersonalizationId">1648548270474-16153</_wcf:ContextData>
      <_wcf:ContextData xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation" name="channelId">-4</_wcf:ContextData>
      <_wcf:ContextData xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation" name="contentPersonalizationId">1648548270474-16153</_wcf:ContextData>
      <_wcf:ContextData xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation" name="channelId">-4</_wcf:ContextData>
      <_wcf:ContextData xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation" name="contentPersonalizationId">1648548270474-16153</_wcf:ContextData>
      <_wcf:ContextData xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation" name="channelId">-4</_wcf:ContextData>
    </_wcf:BusinessContext>
  </oa:ApplicationArea>
  ▼<_inv:DataArea xmlns:_inv="http://www.ibm.com/xmlns/prod/commerce/9/inventory">
   ▼<oa:Get xmlns:oa="http://www.openapplications.org/oagis/9">
      <oa:Expression xmlns:oa="http://www.openapplications.org/oagis/9" expressionLanguage="_wcf:XPath">
      {_wcf.ap=IBM_Store_Details}/InventoryAvailability[InventoryAvailabilityIdentifier/ExternalIdentifier[CatalogEntryIdentifier/ExternalIdentifier[(PartNumber='102
      and AvailableQuantity[(@uom='C62')] and (InventoryAvailabilityIdentifier/ExternalIdentifier[OnlineStoreIdentifier/ExternalIdentifier[(NameIdentifier='Karkkaine
      or PhysicalStoreIdentifier[(ExternalIdentifier='')]]])]</oa:Expression>
    </oa:Get>
  </_inv:DataArea>
</_inv:GetInventoryAvailability>
```

Figure 20. Request sent to OM by site

Comparing the two types of SOAP messages, it can be noted that there is a significant difference in the structure of both the requests and responses between the site and the OM systems. The field names and values are different and the OM’s message contains more information regarding an inventory item. To solve this, there should be some kind of translation between the messages when the OMS Simulator is used to solve the display of inventory data in the development environment.

Just like the request, the response (Figure 21) is also hugely different compared to the messages used by OMS Simulator. This compatibility issue is needed to be solved to integrate the OMS Simulator into the development workflow seamlessly and to show the inventory data inside that environment. As it can be seen in Figure 21 the response contains different fields which have the “ContextData” tag. These contain significantly more information compared to the message of the OMS Simulator regarding the currency, locale, and language. Also, content personalization IDs are transmitted, holding information about the customer based on their activity history (like browsing or earlier orders).



```
▼<_inv:ShowInventoryAvailability xmlns:_cor="http://WCToSSFSMediationModule/correlation"
 xmlns:_inv="http://www.ibm.com/xmlns/prod/commerce/9/inventory" xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
 xmlns:oa="http://www.openapplications.org/oagis/9" xmlns:scwc="http://www.sterlingcommerce.com/scwc/"
 xmlns:udt="http://www.openapplications.org/oagis/9/unqualifieddatatypes/1.1" xmlns:xalan="http://xml.apache.org/xslt"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  ▼<_wcf:ApplicationArea>
     <oa:CreationDateTime xsi:type="udt:DateTimeType">2022-03-29T14:15:24Z</oa:CreationDateTime>
   </_wcf:ApplicationArea>
  ▼<_inv:DataArea>
    ▼<_inv:InventoryAvailability>
      ▼<_inv:InventoryAvailabilityIdentifier>
        ▼<_wcf:ExternalIdentifier>
          ▼<_wcf:CatalogEntryIdentifier>
            ▼<_wcf:ExternalIdentifier>
                <_wcf:PartNumber>102026586</_wcf:PartNumber>
              </_wcf:ExternalIdentifier>
            </_wcf:CatalogEntryIdentifier>
          ▼<_wcf:OnlineStoreIdentifier>
            ▼<_wcf:ExternalIdentifier>
                <_wcf:NameIdentifier>KarkkainenESite</_wcf:NameIdentifier>
              </_wcf:ExternalIdentifier>
            </_wcf:OnlineStoreIdentifier>
          </_wcf:ExternalIdentifier>
        </_inv:InventoryAvailabilityIdentifier>
        <_inv:InventoryStatus>Available</_inv:InventoryStatus>
        <_inv:AvailableQuantity uom="C62">21.00</_inv:AvailableQuantity>
      ▼<_wcf:UserData>
          <_wcf:UserDataField name="customField3">2022-03-31T00:00:00.000+03:00</_wcf:UserDataField>
        </_wcf:UserData>
      </_inv:InventoryAvailability>
    </_inv:DataArea>
  </_inv:ShowInventoryAvailability>
```

Figure 21. Response sent by OM to site

6.2 Solution 1: Using a mediator with provided mediation module

6.2.1 Introduction

The first possible solution is about creating a mediation module for transforming and translating the SOAP messages in a way that the OMS Simulator and the site are able to recognize and digest the information that the XML carries. By doing so, the compatibility problem can be solved without altering any source code on both sides and creating a solution that can be tested separately from the standalone modules in the project.

6.2.2 Dataflow

In the following figure (Figure 22), the dataflow of this type of solution can be seen. Compared to Figure 7 there is a module between the UI and the OMS Simulator which continuously listens for messages from both sides. When a message is received, it translates the message based on which direction did it come and forwards it to the other side. It carries the same data but in another format complying with the required structure by the specification of the currently used messages on the test and development environment.

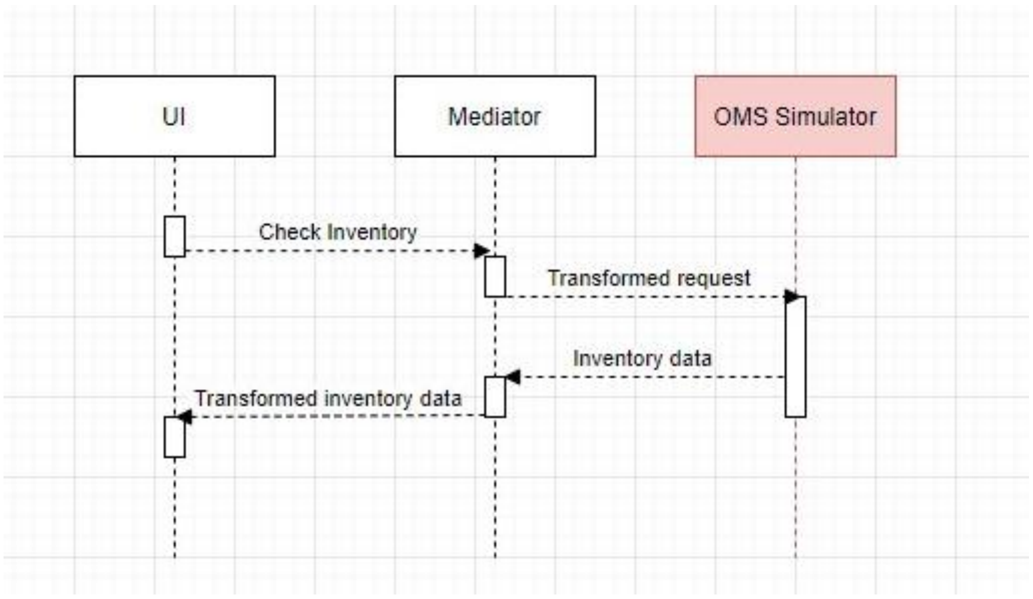


Figure 22. Dataflow when using a mediator module

6.2.3 Mediation module

According to the documentation provided by IBM and HCL for version 8 of the used software, there is a possibility to use a mediation module that helps to translate the messages between the site and the simulator.

The following steps should be completed to make the mediation module work:

In WebSphere Commerce Developer, the WebSphere Commerce test server needs to be up and running, while a new workspace has to be also created for setting up the module. Import the ExtOMSSimMediationModule.zip file as a project interchange into the workspace.

Import the Foundation-Core.jar and Foundation-Server.jar files by doing the following: right-click the ExtOMSSimMediationModule project. Click Properties > Java EE Module Dependencies. Set the project as a dependent Java project. In the Dependencies section of the mediation module project, select the following projects: Foundation-Core.jar, Foundation-Server.jar, then click the OK button.

Go to the Business Integration view and expand ExtOMSSimMediationModule. Open the Assembly Diagram. In the diagram, right-click the ExtOMSSim and select Show in Properties. In the Properties view, select the Binding tab and change the endpoint address to the following URL: http://hostname:9980/ExtOMSSimWeb/services/ExtOMSSim

Save the changes, then select the Servers view. Right-click WebSphere ESB Server 6.1 and select the Start button. After the server starts and synchronizes, right-click the server and select Add and Remove Projects. Add the mediation module to the server by selecting ExtOMSSimMediationModuleApp and clicking Add button. Click Finish and wait for the mediation module to publish and the server to synchronize.

After completing the following steps, the mediation module should be up and running while translating the messages between the UI and the Simulator.

To complete this method a specific software product called WebSphere Integration Developer is needed. However, Solteq does not have that particular product in the technology stack of this project and does not plan to obtain it due to underutilization and efficiency reasons. Therefore, another solution is needed for this project, but it is worth to note this is the easiest option regarding development and integration. Nevertheless, on the downside the benefits do not outweigh the necessary cost.

6.3 Solution 2: Building a custom mediator using Apache Camel

6.3.1 Introduction

The basic concept behind this solution is similar to Solution 1 as both the dataflow and the business logic are the same. The main difference is the use of a custom-built mediation module by using Apache Camel and Spring Boot. In the following paragraphs the primary technologies are introduced as well as a custom-built starting project is presented with the feature of generating Java classes from the WSDL file provided by the OMS Simulator.

6.3.2 Technologies

Apache Camel is a Java object-based implementation of the Enterprise Integration Patterns that uses an application programming interface to specify routing and mediation rules. It is a rule-based routing and mediation engine for message-oriented middleware software that is free and open-source (Apache, 2022). Connectivity to a wide range of transports and APIs makes integration easy as at both ends, all that is required is the specification of the proper endpoints. Camel is extensible, so it is simple to add new endpoints in the future.

Apache Camel works directly with any type of transport or message mechanism, including HTTP, as well as pluggable Components and Data Format choices using URIs. Apache Camel is a simple library with few dependencies that can be easily integrated into any Java program. Regardless of which kind of transport is utilized, Apache Camel can function with the same API. This implies that the API only has to be learned once, after which it is simple to interact with all of the Components that come pre-installed. Apache Camel supports Bean Binding and integrates well with popular frameworks such as Spring. Camel also offers a lot of help with unit testing the routes an application uses during data transfer (Apache, 2022).

The Java Spring Framework is a well-known open-source enterprise framework for creating standalone, production-ready Java Virtual Machine applications. Spring Boot is a solution that combines three essential features: an opinionated approach to configuration, autoconfiguration, and the ability to build independent apps to accelerate and simplify the development of web applications and microservices using Spring Framework. These capabilities combined to provide a tool that allows to build up a Spring-based application quickly and easily (IBM, 2022).

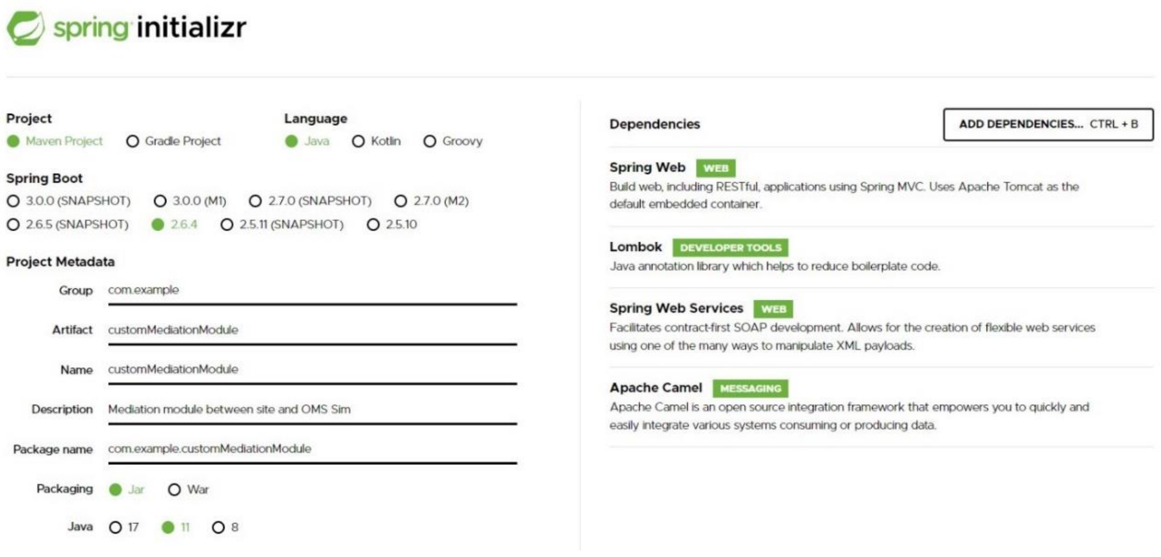
In this solution, Spring Boot and Camel are used together to create a module that accepts SOAP messages and transforms them according to their destination. Camel supports Spring Boot with auto-configuration and starters for a variety of Camel components, but the transformation of messages requires a good amount of development time. Due to time limitations, only setting up the Camel environment and catching SOAP messages with the help of the WSDL file was done during this work. However, this can create a perfect basis for future work if this solution is chosen by the company to implement OMS Simulator.

Apache CXF is an open-source service framework for designing and developing services using frontend programming APIs. These services may interact using a variety of protocols, including XML, HTTP, SOAP and RESTful HTTP, and can transmit data via HTTP or JMS (Apache, 2022). It also supports the use of Spring framework, so it is suitable to use in this work as it helps to solve the SOAP communication between the endpoints.

Maven is a Java-specific build automation tool that assists in the download of dependencies, such as libraries or JAR files. Since there may be different versions of different packages, the tool helps in locating the appropriate JAR files for each project, making the development process easier and faster.

6.3.3 Creating a starting project

Spring Initializr (Figure 23) helped to generate a Spring Boot project by adding the necessary dependencies to start. During this solution, Maven was used for package management and Java 11 is the base version for the application. For dependencies Camel was also added along with Apache CXF, which helps to use and translate the SOAP messages.



spring initializr

Project
[✓ Maven Project] Gradle Project

Language
[✓ Java] Kotlin Groovy

Spring Boot
3.0.0 (SNAPSHOT) 3.0.0 (M1) 2.7.0 (SNAPSHOT) 2.7.0 (M2)
2.6.5 (SNAPSHOT) [✓ 2.6.4] 2.5.11 (SNAPSHOT) 2.5.10

Project Metadata

Group	com.example
Artifact	customMediationModule
Name	customMediationModule
Description	Mediation module between site and OMS Sim
Package name	com.example.customMediationModule
Packaging	[✓ Jar] War
Java	17 [✓ 11] 8

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

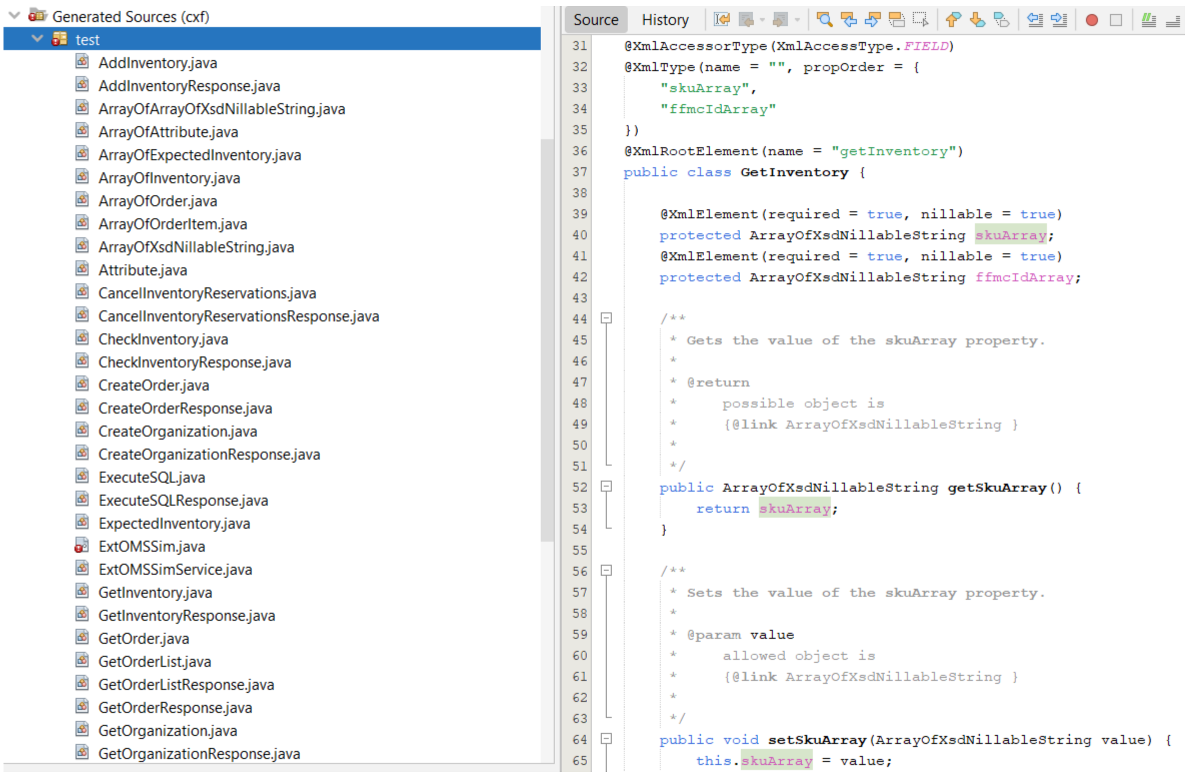
Spring Web Services WEB
Facilitates contract-first SOAP development. Allows for the creation of flexible web services using one of the many ways to manipulate XML payloads.

Apache Camel MESSAGING
Apache Camel is an open source integration framework that empowers you to quickly and easily integrate various systems consuming or producing data.

Figure 23. Using Spring Initializr

6.3.4 Generating classes with Apache CXF

The Apache CXF was configured in the pom.xml to read the WSDL file from the OMS Simulator and process it by creating corresponding classes. Project Object Model (POM) provided project and configuration information that was used by Maven to create the project, which was described in pom.xml. After running the project the corresponding Java classes were generated, so the WSDL file was successfully loaded and processed. The classes and the content of the GetInventory.java file (which was created from the GetInventory message) can be seen in Figure 24.



```
Generated Sources (cxf)
  test
    AddInventory.java
    AddInventoryResponse.java
    ArrayOfArrayOfXsdNillableString.java
    ArrayOfAttribute.java
    ArrayOfExpectedInventory.java
    ArrayOfInventory.java
    ArrayOfOrder.java
    ArrayOfOrderItem.java
    ArrayOfXsdNillableString.java
    Attribute.java
    CancelInventoryReservations.java
    CancelInventoryReservationsResponse.java
    CheckInventory.java
    CheckInventoryResponse.java
    CreateOrder.java
    CreateOrderResponse.java
    CreateOrganization.java
    CreateOrganizationResponse.java
    ExecuteSQL.java
    ExecuteSQLResponse.java
    ExpectedInventory.java
    ExtOMSSim.java
    ExtOMSSimService.java
    GetInventory.java
    GetInventoryResponse.java
    GetOrder.java
    GetOrderList.java
    GetOrderListResponse.java
    GetOrderResponse.java
    GetOrganization.java
    GetOrganizationResponse.java
```

```java
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "skuArray",
    "ffmcIdArray"
})
@XmlRootElement(name = "getInventory")
public class GetInventory {

    @XmlElement(required = true, nillable = true)
    protected ArrayOfXsdNillableString skuArray;
    @XmlElement(required = true, nillable = true)
    protected ArrayOfXsdNillableString ffmcIdArray;

    /**
     * Gets the value of the skuArray property.
     *
     * @return
     *     possible object is
     *     {@link ArrayOfXsdNillableString }
     *
     */
    public ArrayOfXsdNillableString getSkuArray() {
        return skuArray;
    }

    /**
     * Sets the value of the skuArray property.
     *
     * @param value
     *     allowed object is
     *     {@link ArrayOfXsdNillableString }
     *
     */
    public void setSkuArray(ArrayOfXsdNillableString value) {
        this.skuArray = value;
```

Figure 24. Generated classes by Apache CXF

Now the application can create and receive SOAP messages containing elements similar to the structure described in the WSDL file. If this solution is decided to be followed during the practical implementation of the OMS Simulator this application provides a solid base to continue on this path with development.

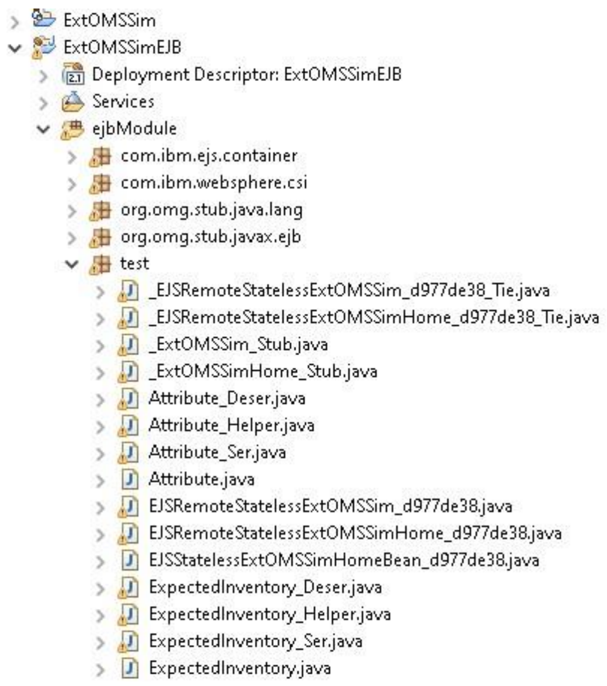
6.4 Solution 3: Rewriting the code of the OMS Simulator

6.4.1 Introduction

After setting up the simulator from the compressed enterprise archive (EAR) file, its source code can be observed and even modified. Through rewriting the part of the code where the message is constructed the compatibility issue could be solved.

After the EAR file was added to the project three packages were imported for running the simulator: ExtOMSSim, ExtOMSSimEJB and ExtOMSSimWeb. The ExtOMSSimEJB contains the business logic of creating the response messages for the requests and putting together the

XML structures by populating the different properties with data. In Figure 25, under the package named 'test' a similar structure can be seen as in Figure 24, which was created by Apache CXF from the WSDL file.

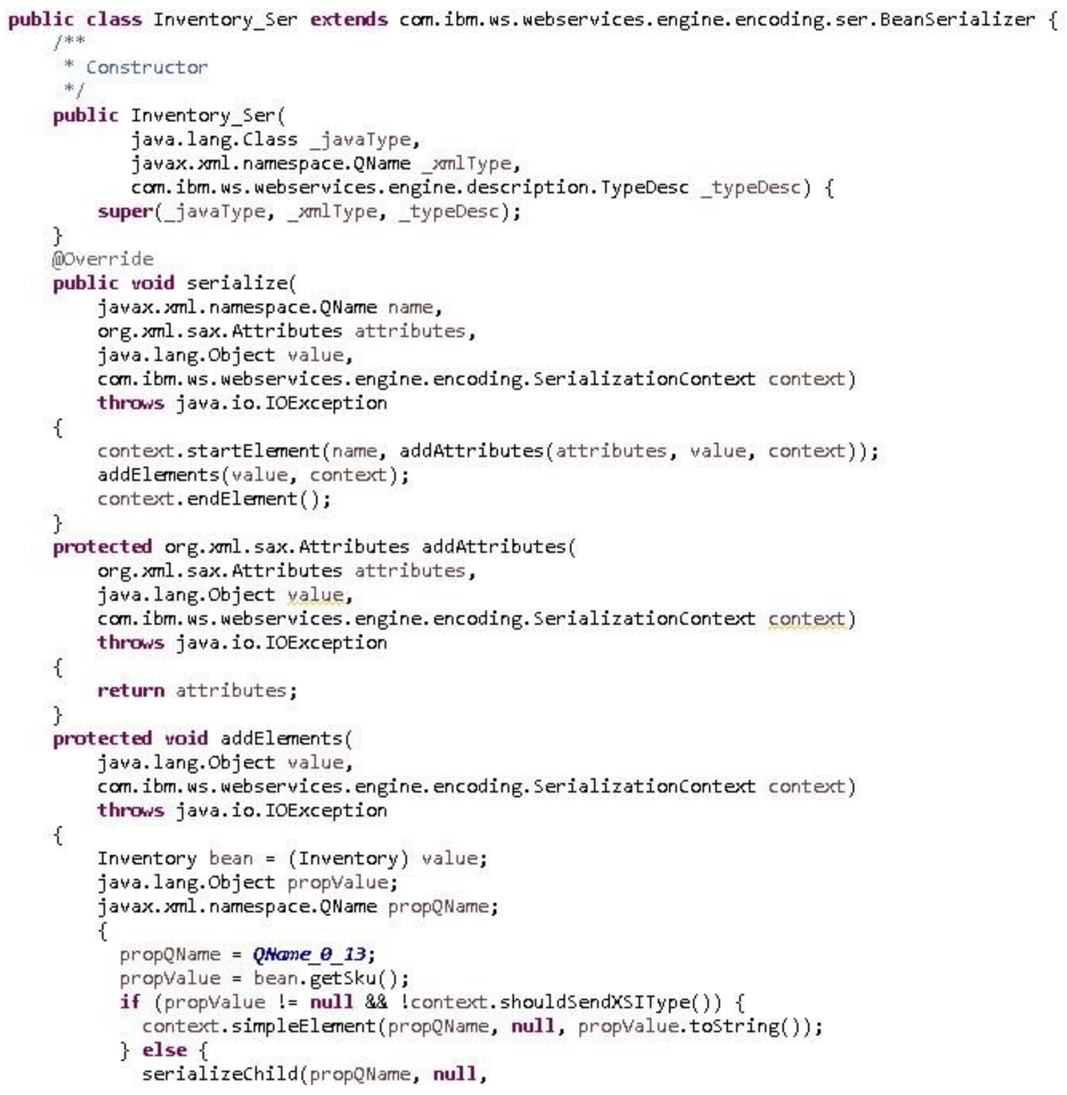


```
> ExtOMSSim
v ExtOMSSimEJB
    > Deployment Descriptor: ExtOMSSimEJB
    > Services
    v ejbModule
        > com.ibm.ejs.container
        > com.ibm.websphere.csi
        > org.omg.stub.java.lang
        > org.omg.stub.javax.ejb
        v test
            > _EJSRemoteStatelessExtOMSSim_d977de38_Tie.java
            > _EJSRemoteStatelessExtOMSSimHome_d977de38_Tie.java
            > _ExtOMSSim_Stub.java
            > _ExtOMSSimHome_Stub.java
            > Attribute_Deser.java
            > Attribute_Helper.java
            > Attribute_Ser.java
            > Attribute.java
            > EJSRemoteStatelessExtOMSSim_d977de38.java
            > EJSRemoteStatelessExtOMSSimHome_d977de38.java
            > EJSStatelessExtOMSSimHomeBean_d977de38.java
            > ExpectedInventory_Deser.java
            > ExpectedInventory_Helper.java
            > ExpectedInventory_Ser.java
            > ExpectedInventory.java
```

Figure 25. Structure of ExtOMSSimEJB package

After observing the source code of the OMS Simulator files it can be seen that the Enterprise JavaBeans (EJB) are used to encapsulate the business logic of the application. For the development and deployment of component-based enterprise applications, EJB is the server-side component architecture. Based on Java EE technology, this technology allows for the quick and easy creation of portable, transactional and distributed applications (Oracle, 2022).

In Figure 26 part of the Inventory_Ser class can be seen. It extends the BeanSerializer class, which helps with the serialization of the data which carries the information for the SOAP messages and generates encoded SOAP messages. There are also classes for deserialization (for example: Inventory_Deser.java), which is the opposite serialization process. Its purpose is to get the data from SOAP messages.



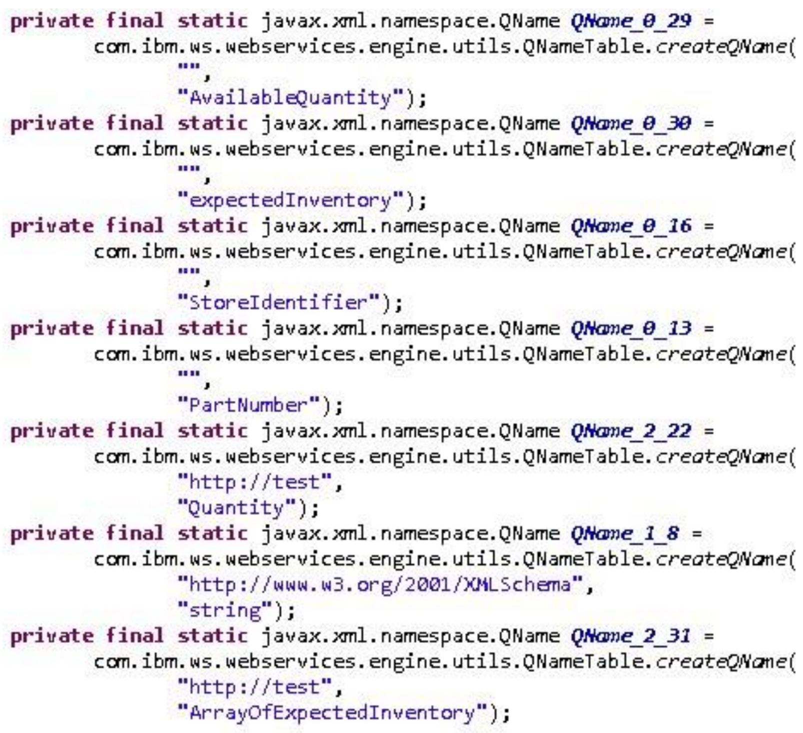
```
public class Inventory_Ser extends com.ibm.ws.webservices.engine.encoding.ser.BeanSerializer {
    /**
     * Constructor
     */
    public Inventory_Ser(
           java.lang.Class _javaType,
           javax.xml.namespace.QName _xmlType,
           com.ibm.ws.webservices.engine.description.TypeDesc _typeDesc) {
        super(_javaType, _xmlType, _typeDesc);
    }
    @Override
    public void serialize(
        javax.xml.namespace.QName name,
        org.xml.sax.Attributes attributes,
        java.lang.Object value,
        com.ibm.ws.webservices.engine.encoding.SerializationContext context)
        throws java.io.IOException
    {
        context.startElement(name, addAttributes(attributes, value, context));
        addElements(value, context);
        context.endElement();
    }
    protected org.xml.sax.Attributes addAttributes(
        org.xml.sax.Attributes attributes,
        java.lang.Object value,
        com.ibm.ws.webservices.engine.encoding.SerializationContext context)
        throws java.io.IOException
    {
        return attributes;
    }
    protected void addElements(
        java.lang.Object value,
        com.ibm.ws.webservices.engine.encoding.SerializationContext context)
        throws java.io.IOException
    {
        Inventory bean = (Inventory) value;
        java.lang.Object propValue;
        javax.xml.namespace.QName propQName;
        {
          propQName = QName_0_13;
          propValue = bean.getSku();
          if (propValue != null && !context.shouldSendXSIType()) {
            context.simpleElement(propQName, null, propValue.toString());
          } else {
            serializeChild(propQName, null,
```

Figure 26. Part of the simulator’s source code

6.4.2 Development

The purpose of this development work was to show the possibility of modifying the source code of the OMS Simulator provided by HCL and IBM in a way that the messages and properties can be altered to meet the requirements of the OM messages, which the site expects to digest and process.

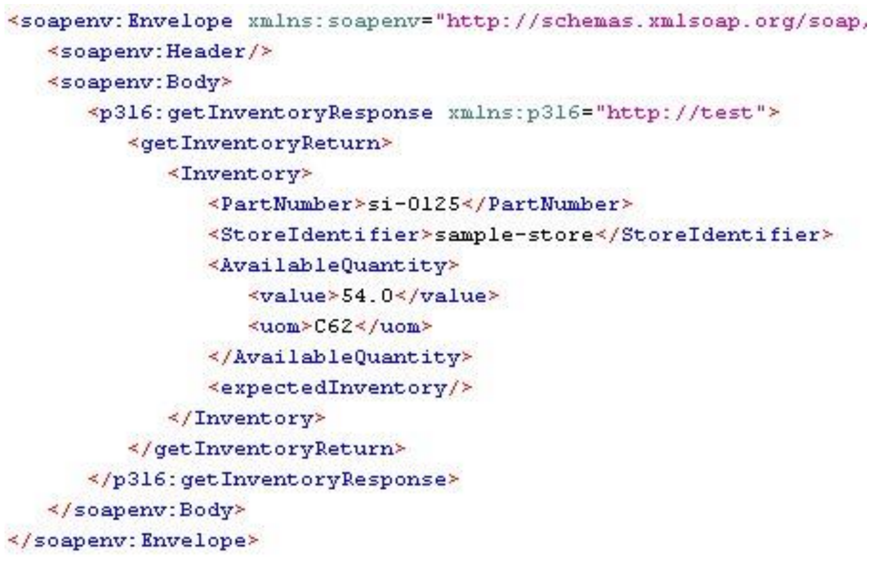
The first task was to decide which command should be used for this process. It was essential to choose a message which is similar to a frequently used OM message. During this work the earlier introduced GetInventory message was chosen. For implementation, the part for serialization of this message was modified by changing the property names and structure in the corresponding class, which is responsible for the GetInventory message. In Figure 27 part of the modified code can be seen in the Inventory_Ser.java file.



```
private final static javax.xml.namespace.QName QName_0_29 =
          com.ibm.ws.webservices.engine.utils.QNameTable.createQName(
                    "",
                    "AvailableQuantity");
private final static javax.xml.namespace.QName QName_0_30 =
          com.ibm.ws.webservices.engine.utils.QNameTable.createQName(
                    "",
                    "expectedInventory");
private final static javax.xml.namespace.QName QName_0_16 =
          com.ibm.ws.webservices.engine.utils.QNameTable.createQName(
                    "",
                    "StoreIdentifier");
private final static javax.xml.namespace.QName QName_0_13 =
          com.ibm.ws.webservices.engine.utils.QNameTable.createQName(
                    "",
                    "PartNumber");
private final static javax.xml.namespace.QName QName_2_22 =
          com.ibm.ws.webservices.engine.utils.QNameTable.createQName(
                    "http://test",
                    "Quantity");
private final static javax.xml.namespace.QName QName_1_8 =
          com.ibm.ws.webservices.engine.utils.QNameTable.createQName(
                    "http://www.w3.org/2001/XMLSchema",
                    "string");
private final static javax.xml.namespace.QName QName_2_31 =
          com.ibm.ws.webservices.engine.utils.QNameTable.createQName(
                    "http://test",
                    "ArrayOfExpectedInventory");
```

Figure 27. Modified source code of OMS Simulator

After the development was finished and the servers were restarted, the newly created code inside the IBM-provided OMS Simulator processed the SOAP message and created the new response, which can be observed in Figure 28.



```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap,
   <soapenv:Header/>
   <soapenv:Body>
      <p316:getInventoryResponse xmlns:p316="http://test">
         <getInventoryReturn>
            <Inventory>
               <PartNumber>si-0125</PartNumber>
               <StoreIdentifier>sample-store</StoreIdentifier>
               <AvailableQuantity>
                  <value>54.0</value>
                  <uom>C62</uom>
               </AvailableQuantity>
               <expectedInventory/>
            </Inventory>
         </getInventoryReturn>
      </p316:getInventoryResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Figure 28. SOAP response after the code modification

The result of this prototype was a successful modification of the IBM and HCL provided OMS Simulator's source code by changing the properties to meet the requirements for OM messages making the messages sent by the simulator similar to the messages which are currently used in test and production environment. Now it is proven that the source code can be altered, and this could be a possible solution to make the simulator compatible with the existing systems and formats used in the project.

6.5 Reviewing the solutions

Reviewing all the three solutions, both advantages and disadvantages can be observed. The least time-consuming solution is to use the mediation module provided by HCL Commerce as the mediator between the site and the OMS Simulator. However, to use this solution a specific software product called WebSphere Integration Developer is needed, which is owned by IBM. To obtain the license for this software is costly and a complicated process so another solution is needed for this particular project. However, it is good to note that this possibility can be useful for other projects where the same issue exists.

In contrast, building a custom mediation module could solve this issue that would operate on a similar principle to the solution discussed above. The positive side of this approach is the use of Apache Camel, which is open-source software, and there are thousands of ready-to-use packages that can help process the SOAP messages and the communication. The downside is that it is time-consuming as the translations have to be done by code written from scratch.

The third and last solution introduced in this thesis uses a different approach. It is based on rewriting the OMS Simulator's existing Java code and solving the compatibility issues without any mediation module. Similar to Solution 2, this approach is also time-consuming, but no new software is needed to run in the background.

All in all, these advantages and disadvantages are needed to be considered when the final decision is made about which would be implemented inside the project environment.

7 Conclusion

The purpose of this work was to find answers and solutions to the question of how the OMS system could be simulated or implemented in the development environment for one of the projects at Solteq, so the local testing of new features and lines of codes could be easier and more reliable before deploying it to test or development environment. One of the main impediments to completing this task was solving the compatibility issue between the OM messages sent by the website and the messages sent by the OMS simulator. This work also tried to offer alternative solutions to this question while summarizing the possible advantages and disadvantages.

As this was the first eCommerce project it was challenging to start the work as the basic concepts, systems, and workflow had to be understood before digging deeper in the subsystems and understand the connections between them. During the first month, familiarization with development tools, systems, databases, and dataflows was done. After that, the problem and the current state were assessed, and the work started by searching for solutions to solve the main issue of this project. While researching the existing architecture and dataflow, new ways of possible implementations were discovered by using the existing toolkits and systems.

This thesis work went through the possible solutions and implementations for solving this issue while introducing the basic concepts and trends in the field of eCommerce. As a result of looking for answers and doing research, three possible solutions were presented: using a mediation module between the eCommerce site and the HCL provided OMS Simulator with WebSphere Integration Developer, creating a custom mediation module with Camel and Spring Boot, and altering the source code of the OMS Simulator to solve the compatibility issue between the components. Furthermore, the reader was familiarised with how to set up the IBM-provided OMS Simulator on an existing WebSphere Application Server, while the technology in running in the background was also presented.

With this work, the author has a well-established knowledge of working with eCommerce while also understanding the structures and processes which are connected to an HCL website and the messages between OM, the OMS Simulator and the website.

The result of the project is a collection of possible solutions, while those ideas are also analyzed by time sensitivity and complexity. With this work, the team working on this project at Solteq can select the most efficient idea. Furthermore, the author and his teammates can continue the implementation phase while having a starting point of solutions using the findings and project files of this work in the near future.

References

Adobe. (2021). *E-commerce platforms*. Retrieved February 14, 2022, from adobe.com: https://business.adobe.com/au/glossary/ecommerce-platforms.html

Apache. (2022). *Apache Camel Product Documentation*. Retrieved February 10, 2022, from camel.apache.org: https://camel.apache.org/manual/faq/what-is-camel.html

Bloomenthal, A. (2020). *Electronic Commerce (Ecommerce)*. Retrieved April 28, 2022, from investopedia.com: https://www.investopedia.com/terms/e/ecommerce.asp

Datanyze. (2022). *E-Commerce Platforms Market Share*. Retrieved January 28, 2022, from datanyze.com: https://www.datanyze.com/market-share/e-commerce-platforms

Honaman, J. (2021). *Headless Commerce*. Retrieved March 21, 2022, from aws.amazon.com: https://aws.amazon.com/blogs/industries/headless-commerce-what-is-it-and-why-does-it-matter-to-cpgs/

IBM. (2022). *Java Spring Boot*. Retrieved April 6, 2022, from ibm.com: https://www.ibm.com/cloud/learn/java-spring-boot

IBM. (2022). *Order Management*. Retrieved March 3, 2022, from ibm.com: https://www.ibm.com/topics/order-management

IBM. (2022). *Product Documentation*. Retrieved March 2, 2022, from HCL Commerce: https://help.hcltechsw.com/commerce/8.0.0/index.html

Insights, C. (2021). *What Is Headless Commerce?* Retrieved March 24, 2022, from cbinsights.com: https://www.cbinsights.com/research/report/what-is-headless-commerce/

Oracle. (2022). *Software documentation*. Retrieved March 3, 2022, from docs.oracle.com: https://docs.oracle.com/cd/E19798-01/821-1796/aeqex/index.html

Skubana. (2019). *Distributed Order Management*. Retrieved January 31, 2022, from skubana.com: https://www.skubana.com/blog/distributed-order-management

SoapUI. (2022). *Working with WSDLs*. Retrieved March 17, 2022, from soapui.org: https://www.soapui.org/docs/soap-and-wsdl/working-with-wsdls/

Solteq. (2022). *Strategy and focus*. Retrieved January 30, 2022, from solteq.com: https://www.solteq.com/en/investors/solteq-as-an-investment/strategy-and-focus

Statista. (2021). *Digital Market Outlook*. Retrieved February 2, 2022, from statista.com: https://www.statista.com/outlook/dmo/ecommerce/worldwide

Voidonicolas, R. (2021). *What is Shopify?* Retrieved February 17, 2022, from shopify.com: https://www.shopify.com/blog/what-is-shopify

WorldWideWebConsortium. (2008). *XML Essentials*. Retrieved March 4, 2022, from w3.org: https://www.w3.org/standards/xml/core

Zidane, O. (2021, November 24). *Global E-Commerce and the Impact of COVID-19*. Retrieved February 15, 2022, from InfoMineo: https://infomineo.com/global-e-commerce-and-the-impact-of-covid-19/