INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Geração Automática de Ecrãs Dinámicos

JOSÉ MIGUEL DOS SANTOS PINHO
Julho de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Automated Dynamic Screen Generation

José Miguel Santos Pinho

Mestrado em Engenharia de Software

Supervisor: Doutor Paulo Gandra de Sousa

Judge:

President:

Vogals:

Porto, 2020

To my family

...

Resumo

Nos dias de hoje, um sistema informático que faça uso de uma vertente web e se enquandre no setor segurador, tem a necessidade de obter informação através da comunicação com sistemas externos tais como sistemas de definição de produto que são responsáveis por modelar produtos de seguros, obtendo assim metadados que, uma vez interpretados, fazem com que o ecrã que se pretende que se renderize seja diferente em função da interpretação destes metadados. Por vezes, a volatilidade dos metadados interpretados pode ser tão grande fazendo com haja a necessidade de re-renderizar todo o ecrã em função destes.

Pretende-se assim dar apoio ao trabalho desenvolvido pela empresa msg life Iberia cujo produto é uma plataforma denominada de msg.Sales utilizada para a comercialização de seguros de diferentes ramos do setor segurador. A plataforma msg.Sales faz uso de um sistema responsável por efetuar a modelação de diversos produtos. Este sistema externo é denominado por Product Definition System (PDS) e o output dado por ele é um modelo canónico que representa produtos de seguros, apresentando assim grande volatilidade.

Assim, ao fazer uso normal das soluções web desenvolvidas pela msg life Iberia é feita uma constante interação com o PDS de forma a obter o modelo de um produto de seguros que se pretende renderizar para uma determinada página. Dentro desta mesma página, é possível que o PDS gere vários metamodelos diferentes para a mesma página. Devido a isto, a alteração de um campo no ecrã pode fazer com que sejam adicionados, removidos e alterados campos que são definidos por este metamodelo.

Devido a esta volatilidade, por vezes é necessário efetuar-se novas renderizações de todo o ecrã uma vez que é necessário haver a constante interpretação dos metadados fornecidos pelo PDS o que causa um problema na escalabilidade da solução principalmente ao nível de performance.

O objetivo desta tese é assim desenvolver um protótipo de um gerador de ecrãs que possa ser incorporado no processo de build de uma solução, fazendo uso de uma Domain Specific Language (DSL) que existe já atualmente no sistema msg.Sales denominada de Flow, evitando assim a interpretação total de ecrãs em runtime. O foco é essencialmente em PDS cujos produtos sejam estáticos, tendo assim menor grau de variabilidade na sua estrutura.

Palavras-chave: *Geração de ecrãs, Performance, Product Definition System, Build, Seguros, Metamodelos, Domain Specific Language, Flow*

Abstract

These days, an informatics system which has a web component and is focused on the insurance sector, has the necessity of obtaining data relative to the commercialized insurance products by communicating with external systems such as Product Definition Systems (PDS) obtaining this way metadata that is further interpreted resulting on a screen being rendered accordingly to these metadata. Sometimes, the metadata is volatile enough to make a screen being re-rendered in order to for the screen to represent the metadata.

This work aims to provide aid to everything developed by the company msg life Iberia whose product is a platform named msg.Sales used for insurance commercialization of diverse sectors. The msg.Sales platform uses an external system responsible for modeling the products to be used. This system is named Product Definition System and its output is the canonical model which represents products showcasing this way a high level of volatility.

As stated above, the normal use of the web application has the need of constantly interacting with the PDS to retrieve the metamodel which corresponds to a particular screen. Within this same screen, it is also possible that the PDS generates different metamodels to represent this same screen. As a result of such volatility, within a specific page it is also possible that by changing fields, there could be new fields being added, existing fields being removed, or the domain of possible values for a particular field changing.

Due to this level of volatility, sometimes it is required to re-render a screen entirely since it is necessary to process the metadata supplied by the PDS which causes a problem in the msg.Sales scalability mainly at the performance factor.

This thesis's goal is to develop a screen generator prototype which can be incorporated within the build process of the msg.Sales platform, using a Domain Specific Language (DSL) that is incorporate within msg.Sales system, avoiding this way the interpretation of complete screens at runtime. The goal is essentially in PDS's whose products are static, having this way a smaller level of variability in its products structure.

Keywords: *Screen Generation, Performance, Product Definition System, Build, Insurance, Metamodels, Domain Specific Language, Flow*

Acknowledgments

I would like start by thanking my supervisor Paulo Sousa for giving me the opportunity to work on a very interesting subject while providing constant useful insight that was helpful throughout this thesis.

Secondly, I would like to thank all my family and friends at msg life Iberia for motivating me to achieve my goals and being ready to help at any necessary time.

To everyone that was involved on the realization of this thesis, a very big thank you!

Contents

List of Figures

List of Tables

List of Equations

Abbreviations

PDS	Product Definition System
DSL	Domain Specific Language
EAP	Enterprise Application Platform
ORM	Object Relational Mapper
M2M	Model to Model
DTO	Data Transfer Object
XML	Extensible Markup Language
OCL	Object Constraint Language
PDM	Product Definition Model
BOM	Business Object Model
POJO	Plain Old Java Object
MDE	Model Driven Engineering
XSLT	eXtensible Stylesheet Language for Transformation
CDL	Concept Description Language
TBL	Translation Based Learning
SAVE	Society of American Value Engineers

1. Introduction

This chapter contextualizes the field that this thesis insides providing a description of the problem that is to be to solved focusing on a specific strategy and goal achievements. The final note of this chapter is to detail how this document is structured, which chapters it contains and what they do represent.

1.1. Context

With the evolution of organizations, the need for technology that improves productivity and decision-making processes among their parties has been growing steadily throughout the years. Organizations are seeking digital solutions to help solving their problems with not only real-time access to critical information, but also ways to store information of their business. [1]

Insurance companies often have extremely volatile and complex products whose data structure may change solely by either a simple medical questionnaire answer or an existing pre-insurance. As such, software solutions developed to enhance insurance sales processes must be able to cope with such volatility and adapt Web Application screens accordingly. Msg life Iberia is enhancing their solutions, looking mainly to improve the user experience and front-end adaptability to provide a solution for the embedded constant changes that product metamodels have in the insurance business. A product metamodel in the insurance context consists in an abstract syntax of a modeling language which represents the entities that are expressed in that modeling language.

One of the biggest challenges that insurance companies face on the insurance market focuses essentially on maintaining a high level of performance more specifically at the usage speed of their solutions during the commercialization phase of their products. The agents that work for insurance companies often are paid by the minute, and in order to reduce the inherent costs of the entire process of creating a policy, they aim to optimize the time spent by the agents to achieve an overall improvement on the productivity of the company while reducing the costs.

The products sold by companies on the insurance sector are often followed with high levels of complexity due to their customer's needs. This level of complexity generates a slow-paced execution of the company processes to generate a policy and risk evaluation, and as such, it is necessary that the software solutions act as catalyzers of the processes. Due to the dynamism of the products sold, it is a must for software to be reactive and fast when transmitting insurance models to screens for their users.

For the particular case at msg life Iberia, the platform msg.Sales integrates itself with different Product Definition Systems – PDS which are capable of customizing insurance products accordingly to the needs showcased by the insurance companies. As such, it is necessary to support the dynamism given by the PDS by being able to quickly adapt the screens showcased to the users as there are constant changes to the product models given by the PDS.

It is necessary to provide a layer of abstraction to the customer since they will be able to customize their own application screens. This creates a problem that msg.Sales has to solve in order to keep their dynamism while maintaining the capacity to render screens configured not by IT teams on msg life Iberia, but by customers.

As such, it is necessary to have a system capable of rendering HTML templates based on metadata provided from a PDS during the build process of the system, which can be used directly or indirectly during the process of rendering a screen which occurs within the system whose screen's configuration is made within Flow files. The consequent rendering of screens should be cached in order to improve the performance of the main system where the solution to the problem addressed by this thesis

1.2. Problem

The problem that this thesis addresses is:

How to improve the performance of screen rendering for dynamic screens based on metadata.

The screen generation performance in Product Definition Systems is drastically affected due to the dynamism that a screen might have in response to modifications in the structure of a product. Often this leads to the need of generating the same screen entirely again due to the changes of a product structure which leads to being necessary to reinterpret the metadata provided by the PDS. This means that for instance, depending on the data inserted by the users of the msg.Sales platform, the rest of the screen can be changed entirely, and as stated above, the metadata will need to be interpreted again for the screen to be adapted correctly to the product structure. In case a screen has multiple fields where a simple value change triggers the mechanisms stated above, it will be noticeable a big performance decrease on processing time and metadata to screen transformation of different alternatives of a web page screen. What this means is the question is not about showing or hiding fields within the application, because some of these fields never existed in the first place. These fields only begin to be relevant according to the input that a user provides within the msg.Sales platform, in accordance to the metadata provided by the PDS.

As the msg.Sales platform has its own canonical model, each interaction with the PDS requires transforming the platform's internal model to the correspondent model within the PDS. This is a costly operation, which needs to occur sideways on the interactive level of both solutions. Since it's the PDS role to provide the updated model of a particular product, when changing a field on a screen which can have impact on the product model structure, the need of transforming msg.Sales internal model into the PDS model is raised so that it can be updated accordingly to user input information. When the model is updated within the PDS, the msg.Sales needs to re-interpret this metadata and build its own model again accordingly so that the screen rendering module within the platform can process this new model and transform it to visible screen in accordance to the newly updated product definition model which can be completely different from the previous. As such, as mentioned previously, it is unavoidable to re-interpret this metadata in order for the screens within msg.Sales to showcase this model accordingly. A simple example of this occurance can be the selection of a checkbox within msg.Sales, which triggers the necessity to fill 2 different forms of data. These 2 forms of data were not present within the metadata given from the PDS before the actual checkbox selection. As such, it can't be possibly rendered or interpreted. However, in case of selecting the checkbox, the communication mechanism between msg.Sales and the PDS mentioned above is triggered, and the insurance product model is updated in order to contain the metadata to all these possible changes which then are re-interpreted in order to update a screen in msg.Sales accordingly.

These screens are intended to be configured by an end-user (Client) using a provided Domain Specific Language (DSL) developed by msg life Iberia called Flow. It is essential to maintain the dynamism in msg.Sales where anything can change at any moment and as such, optimize the process of generating dynamic pages which are configured through Flow in order to improve the user experience and improve overall performance of page rendering.

1.3. Goals and Strategy

The goal of this thesis consists in exploring new alternatives in dynamic page rendering based on metadata that is provided from volatile metamodels and build a solution capable of doing so. This rendering mechanism should be possible to incorporate in the build process of an Enterprise Application Platform (EAP), reducing this way the toll of interpreting as much data as possible at runtime.

In order to explore possible solutions capable of addressing msg life Iberia necessities, this thesis aims to further investigate and optimize rendering techniques by:

- Study the up to date techniques involved in dynamic rendering from metamodels.

- Adapt and enhance already explored techniques so that they can be used in the context of msg life Iberia solutions.
- Measure the results in face of the previous solution.
- Optimize the solution and integrate the solution during the build process of an EAP.
- Incorporate within the build process pipeline of an EAP a new step which does handle HTML template rendering
- Develop a new solution capable of developing HTML templates which are defined by an existing Domain Specific Language (DSL) which is the Flow, followed by a Model to Model (M2M) transformation triggered by the adapter that connects msg.Sales to a PDS.
- Enhance the EAP with a caching mechanism to hold HTML templates.

As listed above, it is important to understand how a M2M transformation occurs within the msg.Sales system since it is an essential step in the communication with a PDS which is going to be explored throughout in order to incorporate this mechanism within the proposed solution. Each of these listed topics are studied in the chapter 4 in order to contextualize and explore the different possibilities to achieve each individual goal.

1.4. Document Structure

This document is structured as follows:

- Chapter 2 is a brief context of all the terms and technologies relevant to this thesis field.
- Chapter 3 provides a value analysis considering that this thesis's problem is solved.
- Chapter 4 describes the state of the art in this thesis's field.
- Chapter 5 evaluates the state of art possibilities and proposes a solution-
- Chapter 6 describes the solution overview.
- Chapter 7 evaluates the solution described in chapter 6.
- Chapter 8 concludes the investigation of Dynamic Screen Rendering.
- Chapter 9 has all the references used within this document.

2. Context

In this chapter it is described the major concepts that are important to have insight through the research process as well as concepts that are necessary for the design and implementation stages.

2.1. Insurance

An insurance is an agreement between entities or an object with a company, which dictates that in case that there is specific damage to the entity/object, the cost of repair can be either totally partial or fully refunded depending on the clauses defined on a contract. The insured person, in exchange for such safety coverages, pays the insurance company a premium.

The contract between the insured person and the insurance company is named Insurance Policy. It details the contract defining which coverages the insurance offers for the Insured Person, providing in addition all the information regarding the obligation and rewards between both parties.

To make an insurance contract, it is necessary to have involved an Insurance Company, an Insured Person, Beneficiaries and a Premium Payer. This means that the person being covered by the Insurance Company on a policy is the Insured Person. In many cases, the Insured Person is not the person which pays the premium to maintain the coverages, being that person the Premium Payer.

Insurance companies' growth do not automatically promote finance growth. As referred by [2], to provide future growth it is necessary for an insurance company to generate and maintain enough capital to satisfy regulators as well as to finance its expansion. The profitability of an Insurance Company is critically dependent on its operations and Financial Activities. While the Financial Activities consist in investing the policies' premium, the operating activity consist of insurance operations about selling policies and servicing existing policies.

2.1.1. Life Insurance

A life insurance consists in a contract between an Insurer and a Policy Holder in which the insurance company promises a death benefit that is relative to the premium payment that the insured does. This means that in case of death, there are beneficiaries that will be granted the benefit upon the death of the Insured.

Dependents can be deeply affected by the death of their peers. According to [3], the number of Life Insurances has been steadily growing over the past years, in response to the premature deaths of the main source of income of most families. As such, the purpose of life insurance is to provide financial protection to surviving dependents after the death of an insured whose contract values are usually calculated after analyzing the history of the insured and the risks that is professional occupation provide.

The Life Insurance has three major components that are detailed on its policies:

- ***Death Benefit*** – It is the amount that will be granted to the beneficiaries upon the death of the insured. The Insurance Company guarantees to provide a death benefit amount based on the estimate amount needed by the heirs of the insured. This value required approval by a company's underwriting requirements which evaluate the risk of the contract and the interest gained by the company.

- ***Premium*** - Consist in the value that was calculated by the insurance company and is used named cost of insurance. This amount is highly variable due to the differences of risk between peers as their lifestyle, medical history and occupational hazards highly influence this value. The company will only be obliged to provide the death benefits to the insured in case that the payment of the agreed premium is up to date.

- ***Cash Value*** – It is either a savings account which can be used by the policy holder during the life of the insured, with the amount accumulated to that date. Most of the policies often have restrictions on withdrawals depending on the use that the withdrawn money will have.

2.1.2. *Processes*

When the time to make a life insurance arrives, the internal processes of the contacted Insurance Company are usually extremely complex. As stated by [4], the selection of the best life insurance contract has been a problem confronted by researchers who have offered consumers numerical recipes to solve for the contract of best value or lowest cost.

As stated at [5], from client's perspective, decision of insurance purchase is not easy at all. Due to the differences in policy features, service quality and cost, the whole process of purchasing takes too much time and energy, and even in every single step there are obstacles. To support the decision process, the insurers often sell their products through agents who will then sell the products for a commission.

A basic business process for a life insurance company can be viewed in Figure 1:

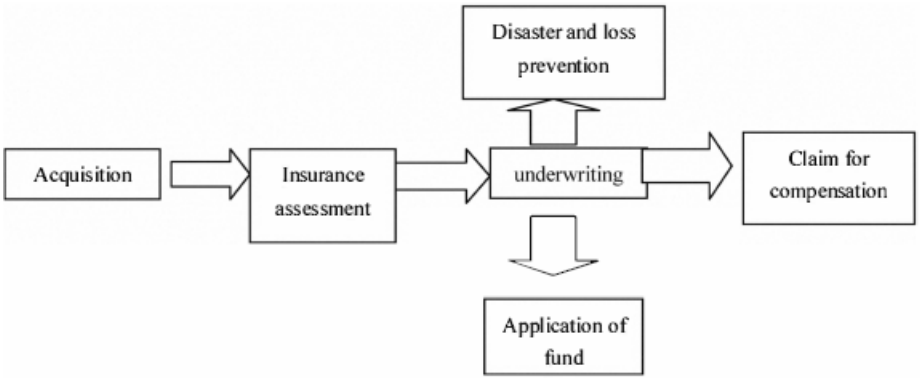

Figure 1 - Life Insurance Business Process from [4]

The business process starts with propagation of an insurance company products and services constantly. Once initial contact is made between the company and the interest party, the initial insurance assessment is started and all relevant data is collected. As premiums are calculated for insurance companies which are derived from the risk that an insurance proves, the insurance company goes through an underwriting process where the risk is evaluated. The final decision of the insurance company on whether they do intend to reach an agreement or not is achieved as soon as the underwriting process concludes.

2.2. msg.Sales

This subchapter provides an insightful representation of who developed msg.Sales, what msg.Sales is and the different external software that have integrations and are relevant to this thesis' field.

2.2.1. msg life Iberia

It is a company that has been contributing to the innovation on the insurance market, looking to digitalize and innovate the processes of the insurance companies. Its goal is to look into the dynamism of the insurance market as an opportunity to provide aid to their future not only by technological aspect of the presented solutions but also in ways where the digital agility allows helping the insurance companies to achieve their strategic and operational goals.

Msg life iberia provides solutions to their clients which cover the basic needs to fulfill the cycle of life of the insurance companies. To do so, msg life iberia has a back-office product which is the product machine, responsible for configuring products and all its rules. As for front-end solution, there is msg.Sales which is capable of integrating with different Product Definition System's such as the Product Machine, focusing on the client and the sales process resulting in scalable and efficient insurance distribution.

2.2.2. Product Definition System

A product definition system is the 3rd party that is integrated with msg.Sales, responsible for providing the definition of each insurance product. Providing a definition is the same as building a complex object structure that represents the product at a point in time. Since insurance products are very dynamic, the constant communication between msg.Sales and the PDS is necessary in order to keep web applications showing the correct data and correct product structure at given times.

The PDS is responsible for:

- Providing the list of all available products
- List of available products per agency
- Define either fully or partially the structure of the product, including all its components and which fields should be in each component.
- Metadata about the fields such as: Mandatory,Read-only, Rating relevant, Minimum, Maximum and accepted Discrete Values.
- Validate a Quote
- Update the metadata of a Quote, taking into account its current status.

Msg.Sales makes use of an adapter to communicate with the PDS's, which usually contain 4 services:

- ***GetProductList*** - Service responsible for obtaining the list of available products
- ***Validate*** - Service responsible for validating a quote at a given time
- ***UpdateFuncionalStates*** - Service responsible for updating the metadata of each component of a quote
- ***BuildAdaptable*** - Service that transforms the metadata provided by the PDS to the internal Business Object Model of msg.Sales

2.2.3. What is msg.Sales

Msg.Sales is a software developed by msg life iberia, and it is responsible for handling the sales and distribution process of insurances in different channels and sectors. In order to achieve its goals, msg.Sales has to be integrated with a Product Definition System (PDS), handling each product as they are defined.

Msg.Sales reuses its base code and components to allow the development of a single dynamic and reusable product, being highly customizable in accord to each client needs'.

To achieve lower coupling between msg.Sales and 3rd party systems adapters are used to connect to specific systems, providing well known interfaces and a model transformer between the 3rd party model and msg.Sales internal model. Figure 2 describes the use of such adapters:

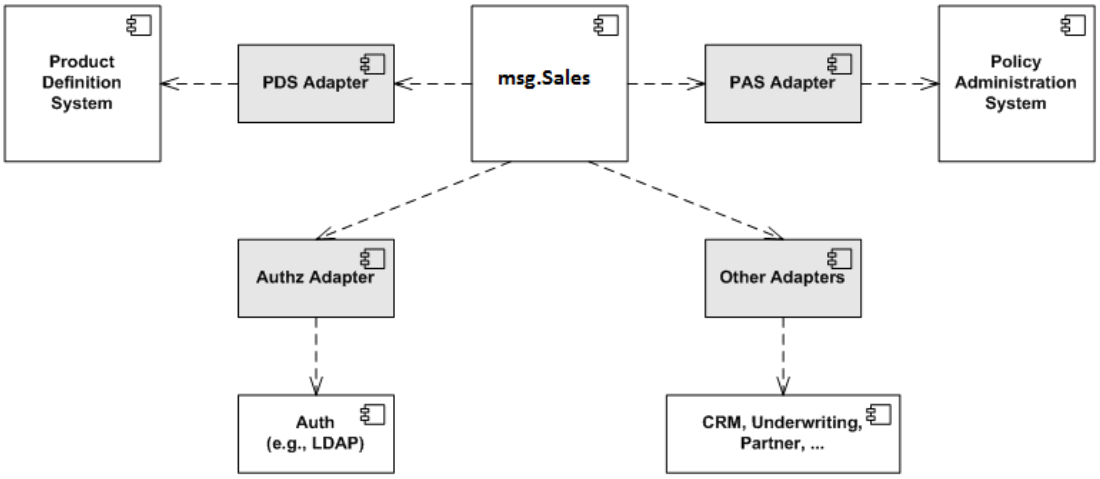

Figure 2 - msg.Sales Architecture [5]

Internally, msg.Sales exposes features through well defined, stable, versioned and documented REST/SOAP endpoints, enabling 3rd parties to integrate themselves with msg.Sales, making use solely of the API without any of the front-end applications.

The figure 3 shows how msg.Sales exposes their services of different modules through REST/SOAP endpoints:

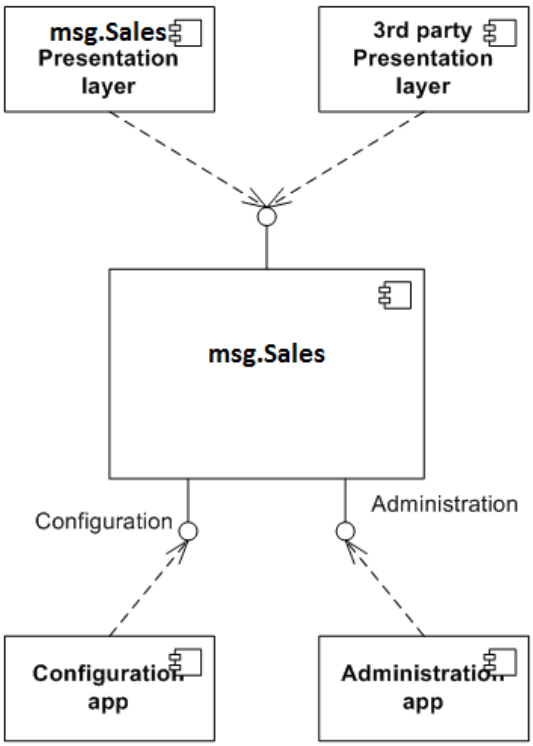

Figure 3 - msg.Sales API and Frontend [5]

In the Figure 4 it is possible to visualize the technological stack used in msg.Sales.

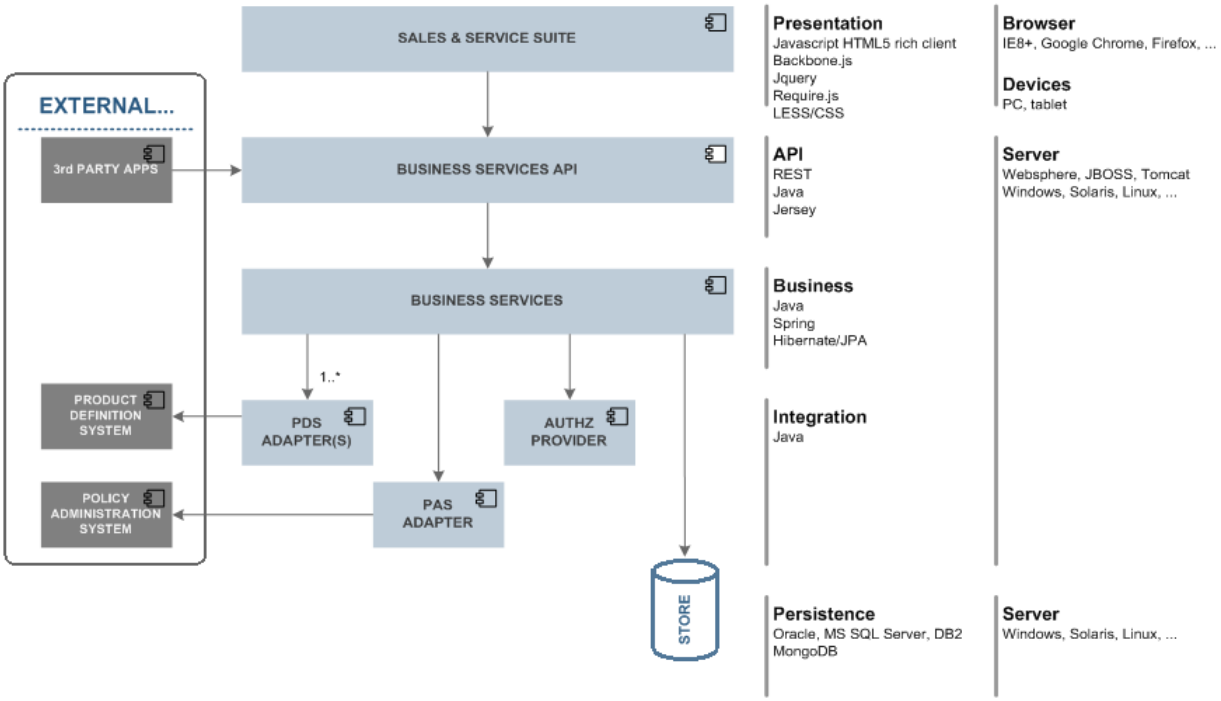

Figure 4 - msg.Sales technological stack [5]

The front-end stack makes use of Backbone.js, JQuery, LESS and Require.js. The msg.Sales API is built upon Java using REST with Jersey and SOAP. Business logic makes use of JAVA with Spring framework and Hibernate as the Object Relational Mapper (ORM).

The front-end layer makes use of a Data Transfer Object (DTO) named container and data holder which contains the current metadata needed to render a screen. The backend is responsible for handling the business logic and updating this DTO accordingly. The metadata that will be included on the container and data holder is provided by a Domain Specific Language (DSL) in the Extensible Markup Language (XML) format, making the front-end highly customizable and dynamic in function of the current backend model data. In order to render the screen, render-queue was built as a framework capable of handling the container and data holder by interpreting its metadata and render HTML in accord to the information passed from the backend to the frontend dynamically.

2.2.4. *Product Machine*

Product Machine is a software that has been developed by FJA-US over the last 20 years. This solution focuses on modeling insurance products in the life, non-life and health fields. Product machine is capable of defining quotes, packages, coverages and calculation rules for every single modeled product. Its main goal is to simplify the process of creating and maintaining products for insurance companies, aiding and optimizing the time spent during development.

The Figure 5 describes a better view of each of the product machine components:

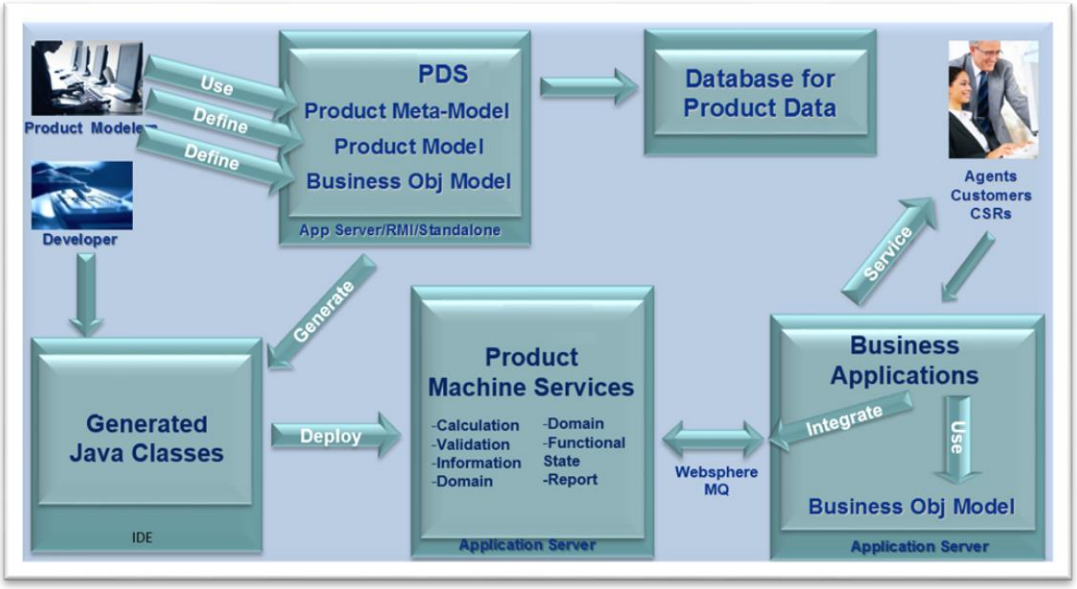

Figure 5 - Product Machine architecture [5]

- ***Product Modelers*** - Consist in the group of people that are responsible for modeling an insurance product, its components, inherent dependencies and business rules.

Product Modelers make use of a workbench named: Modeling Workbench which is a customized eclipse workbench where it is possible to model a product by adding components on the UI. Each product can have a set of business objects, and business objects can have child objects of their type. Each business object holds metadata which is also possible to being configured by the product modeler. The product modeler is able to configure that a specific business object, has a specific attribute which has a data type and value. Business objects can have relations to other business objects, and it is represented with cardinality.

Product Machines metamodels can be described as a Product Definition Model (PDM) since it is a representation of an insurance product model at a given point in time. All the objects/tools that are used in the PDS to model insurance product specifications and model business objects. The product metamodel consists of a class list which contains attributes (data type, field length, trailing decimal spaces, optional, changeable), associations (parent child with cardinality) and constraints (using OCL).

In sum, these are the main advantages of the Product Machine:

- Custom Data Types
- Used to define the type of each class attribute in the meta-model
- Connects to KeyType entities in the PDM (drives valid values in PDM entities)
- Note: Hidden constraint requires a custom data type for each KeyType
- Filter Query Templates
- XML based SQL like statements executed through the filter board
- Enables modelers in the PDM to work with filtered version of entities in PDM

Product Machine makes use of Model-To-Model transformations in order to dynamically generate java classes to represent the product that was modelled with all its characteristics and dependencies. Basically, the object model is generated.

Product machine when deployed, also exposes 7 services, being them:

- ***Calculation*** -> Service that is responsible for calculating the values for premiums and benefits.
- ***Validation*** -> Service that is responsible for enforcing consistency and accuracy
- ***Information*** -> Service that presents available products, options and riders. A rider is the payment of the sum assured on death of the policy holder.
- ***Domain*** -> Service that provides valid values for input fields (possible alternatives)
- ***Functional State*** -> Service that collects all relevant risk information and populates business objects with its attributes (changeable, relevant, master)
- ***Report*** -> Service that constructs and distributes appropriate documents.

Since product machine is built as a JAR, it can be a dependency to business applications. This way, business applications can integrate with PM by the services provided, using the PM business object model.

2.3. Rendering dynamic screens in msg.Sales

Currently, msg.Sales has its own framework to render dynamic screens named Render Queue. The process of rendering a dynamic screen starts by obtaining a Screen Flow Definition provided by the Flow framework. Render Queue is responsible for processing

the metadata that was previously filtered by Flow and transform this metadata into a concrete screen. Some fields within a screen are capable of changing entirely the metadata which is intended to be rendered. As such, changing a field within a screen can trigger the need to render the entire screen again, and as such, process the entire metadata structure again. This metadata structure is fetched from the PDS and consequentially applied a Flow transformation. Render Queue receives once again an entire representation of the current screen metadata, and as such, reinterprets and re-renders the screen as a whole.

2.3.1. Flow

Flow is an internal DSL developed by msg life Iberia which describes how data obtained from product definition systems is placed and shown on the screen.

A domain specific language (DSL) is a language that is design to be used for a specific set of tasks. [6] Affirms that just as a metamodel is a relative term to describe a model that is used as the basis for another model, the term domain-specific is used in a relative sense. The domains are highly variable and usually showcase volatile metadata, being developed to enhance solutions that are contained within a well determined closure.

In [7], it is claimed that it is a high-level software implementation language, which supports concepts and abstractions that are related to a particular domain, being a collection of sentences in either a textual or visual notation followed with particular syntax or semantic rules defined in schema definitions. The contents of the semantic essentially are the definition of metamodel specifications, providing a high-level view abstracting from low-level implementation details.

Some of the advantages on using DSL's mentioned in [8] are reduced time to market, reduced maintenance costs, higher portability, reliability, optimization and testability.

There are two types of flow definitions:

- Navigation Screen Flow -> Describes the sequence of screens and how the navigation is possible between the configured screens. Additionally, the navigation screen flow also allows configuration of the possible movements between each current step, and the necessary application access rights to navigate between each configured step.

- Screen Flow -> This is the main flow artifact; it does describe how the data is placed on a screen.

For every major action in msg.Sales system, there must be a flow definition to define how data appears on the screen. The first definition must always be a Navigation flow definition, even if the action is accomplished with one screen only, there must always be a navigation definition. Each navigation screen flow holds pointers to the screen definition(s) that are possible for a given moment.

Each screen Flow configuration should be mapped with the internal Business Object Model (BOM) in a way that, when interpreted, the configuration for each Plain Old Java Object (POJO) that is intended to be rendered on the front end should point to an existing reference within the BOM.

The basic flow functionality is portrayed in Figure 6:

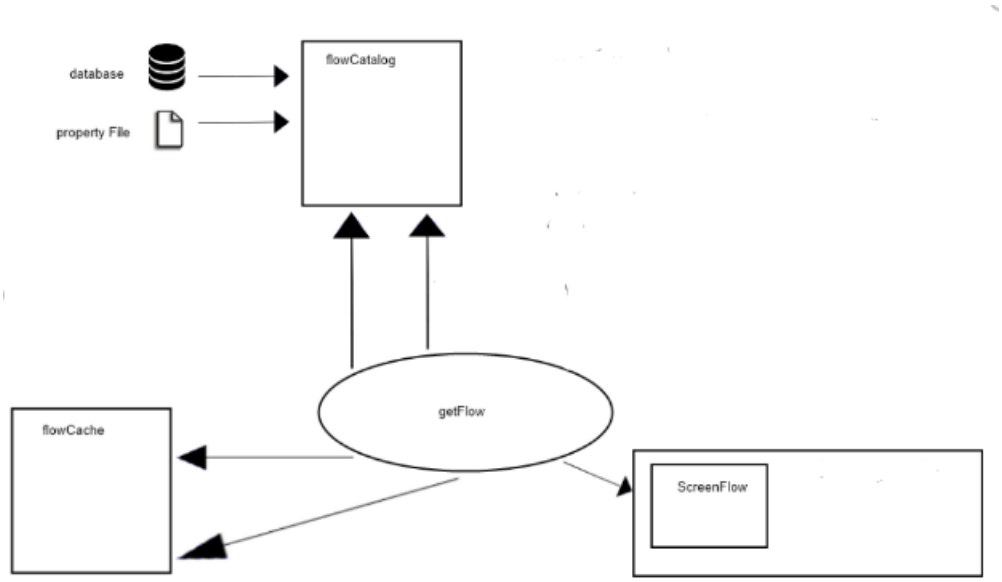

Figure 6 - Flow Functionalities [5]

- ***Flow Catalog*** - This is a catalog that contains all the loaded and available Screen and Navigation flow files.
- ***Flow Cache*** - Cache that saves previously used contexts in order to provide reusability.

The main flow is that the flow catalog is populate by interpreting flow mappings from two possible ways, a property file with the available flow definition files and from a database.

When a screen is intended to be rendered, the system has the need to evaluate which is the correct flow definition for a particular action. That calculation is done by evaluating the current BOM with the key that defines the flow file. The retrieval of the flow file, can be either from the flow cache if the operation is configured to indeed make use of a the flow cache, or retrieved from the process of finding the best screen flow file that has the best score to showcase the screen intended within the current operation.

2.3.2. Render Queue

The frontend of msg.Sales makes use of a sub-system of the msg.Sales frontend framework named Render Queue. This subsystem is responsible to render each screen within the application, interpreting a dynamic metamodel whose information is contained within

- ***Business Object***
- ***Business Property***
- ***Business Attribute***

Each Business Object can have multiple business properties and business attributes. All these entities represented the metadata that is being transferred from the backend to the frontend with the intent of being rendered.

Render queue does accomplish this by queuing all business objects to be rendered, which once the render of a view is dispatch, renders the screen entirely at once.

In order to avoid interacting with the queue interface directly, msg.Sales does make use of a mixin, which has the following functions:

- ***CreateSection*** - Function that creates the needed markup for a section, ready to be populated later by the ***populateSection*** function. Section hints should be supplied within the model to know which kind of fields will get populated inside this section. The render hint would determine which kind of markup will be used to display the attributes.
- ***Dispatch Render*** - Function that does trigger the processes queued within the render queue, leaving it empty once finished.
- ***Render Section*** - Function responsible for rendering a section within a view.
- ***RenderSectionCascade*** - Function that does render a set of sections sequentially.
- ***Populate Section*** - Function that populates a previously created section with the intended metadata.
- ***ApplyBootStrapClasses*** - Function that appends the bootstrap classes to all children elements from a parent element.
- ***RenderTrigger*** - Renders a trigger according to the links defined in the metadata with all needed attributes in order to fire its events.
- ***FromScreenToObject*** - Function that does pick up all the relevant data of a screen, creating a Container and Data Holder to invoke backend services with the current up to date data. A container and Data Holder is a data structure with a model that is known in both the frontend and the backend.

Figure 8 shows the way that normally a screen is rendered within msg.Sales system:

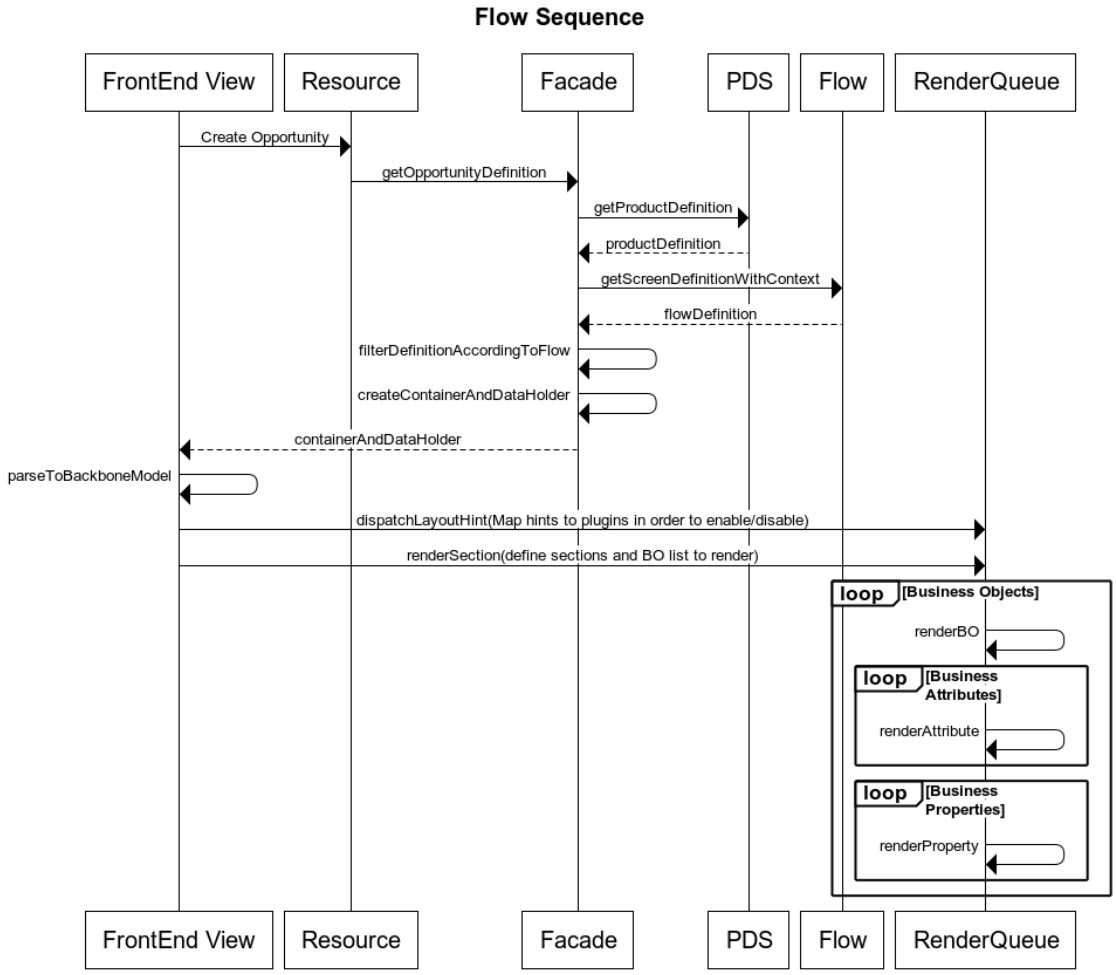

Figure 7 - msg.Sales page rendering process [5]

Every time a screen is rendered, the frontend has sent a request to the services handler which redirects the request to the backend. Once the backend receives the request, he does handle the logic intended to be done within the action that the frontend triggered communicating with the PDS in order to obtain the most up to date metadata according to the action being done. Once the backend has the metadata up to date, he has to send it back to the frontend in order for the screen with the correct data to render. To do so, the backend find the best matched flow file in accord to the action in cause, resulting in the creation of a Container and Data Holder ready to be sent to the frontend whose metadata was filtered according to the flow file matched. The container and data holder information are returned to the frontend, which parses the response to backbone.JS models. The frontend starts interpreting the metadata, looking into the ***layoutHints*** and sections defined on the container and data holder and as a result enables the correct plugins and renders the intended sections for the page that is being rendered. After the render queue is triggered, it starts rendering the container and data holder that is present, starting by recursively rendering all the business objects, business properties and business attributes within the section in context.

2.3.3. Existing Restrictions

Despite Render Queue being highly flexible and capable of rendering metadata that is extremely volatile it does contain some restrictions.

There is the concept of "**Master Field**" on msg.Sales. A Master Field for msg.Sales, is a field that when changed on the screen, do require an update of the metadata, triggering the mechanism described on Figure 8, providing the backend the data changed on the screen by making use of the ***fromScreenToObject*** function. Since it is necessary to convert the Container and Data Holder object that is returned with the most up to date metadata definition to a backbone model and only render queue has knowledge of how to interpret such data and render it, it is necessary to trigger the entire page rendering process resulting on a slower performance within the application and a worse user experience since he would have to wait for the screen to be rendered to make use of the application again.

The biggest restriction of render queue is the one described above, which doesn't allow just the necessary components of the screen to be rendered, but needing to render the entire page again.

Another restriction of the render queue is that rendering customization is only allowed for the callbacks that are known already by the render queue. This means that if for a specific screen, it is necessary to add custom behavior depending on the metadata, it is not intuitive and not easily done making use of the render queue

3. Value Analysis

According to the Society of American Value Engineers (SAVE), “Value analysis is the systematic application of recognized techniques which identify the function of a product or services establish a monetary value for the function and provide the necessary function reliability at that lowest overall cost.”

A value analysis is structured of the following processes:

- ***Functional Analysis*** - Understand why the product is in fact necessary and which the reason for its existence is.
- ***Creatively Analysis*** - Researching different alternatives to solve the problem in cause
- ***Evaluation*** - Evaluate the outcome of the different alternatives and consequentially measure them.

3.1. Selection & Orientation

The most used approach to select and determine orientation within a project is the Paretos’ ABC analysis. It consists in determining the critical areas according to their relative cost and contribution. Using histograms to represent the Pareto’s analysis is suggested in [9] as a tool to enable the identification of dominant variables influencing the cost and the weight each variable holds.

Figure 8 represents an analysis done in function of the existing solution within msg.Sales, and the weight of the areas where improvement is necessary:

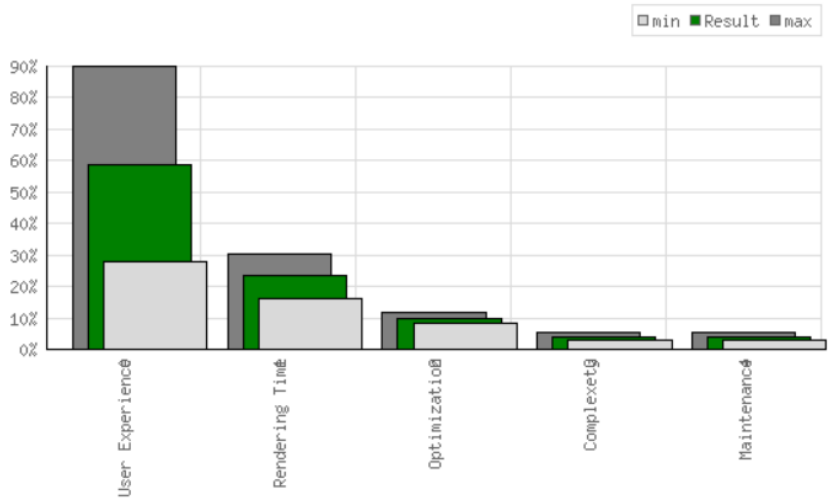

Figure 8 - Pareto's ABC Analysis

The main focus of researching alternative ways of implementing a generic solution has the end goal of providing an engaging and good user experience. Most of this user experience can be achieved by improving the rendering time took for each screen, optimizing a solution to the point where it will have a lower complexity and maintenance expense.

The points defined in Figure 8 represent the weight of each criteria upon the development and research of the dynamic screen rendering project. It is further detailed and explained under Section 3.1.1.

3.1.1. Analytic Hierarchy Process

The Analytic Hierarchy Process (AHP) method is a mean of identifying relevant facts and the interrelationships that exist between them. It consists on a decision-making model that provides aid in making decisions for complex problems. The AHP method has three processes which include identifying and organizing decision objectives, criteria, constraints and alternatives into a hierarchy while evaluating pairwise comparisons between elements at each individual level [10].

To further aid on orientation within this thesis' project and based on the Pareto's ABC Analysis in Section 3.1, an AHP diagram was developed and is showcased on Figure 9:

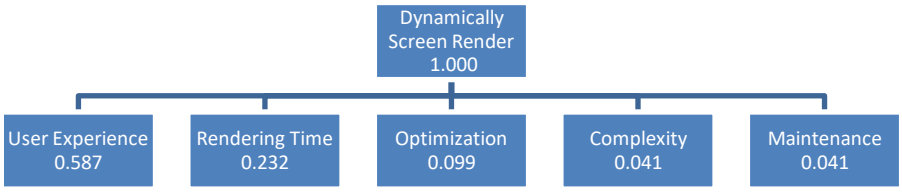

Figure 9 - AHP Decision Hierarchy

To calculate these values, a scale from 1 to 9 (1 is equal importance and 9 is extreme importance) was made in order to pair these factors how compare them against each other, resulting in Table 1:

Table 1 - Criterion Evaluation Table

Factor 1	Factor 2	Prioritization
User Experience	Rendering Time	5
User Experience	Optimization	6
User Experience	Complexity	9
User Experience	Maintenance	9
Rendering Time	Optimization	3
Rendering Time	Complexity	7
Rendering Time	Maintenance	7
Optimization	Complexity	3
Optimization	Maintenance	3
Complexity	Maintenance	1

By analyzing the data and the prioritization a comparison matrix is made between factor comparisons, where if Factor 1 is prioritized as number 5 over Factor 2, the inverse is applied for the Factor 2 prioritization. This process results in the matrix table represented in Table 2:

Table 2 - Decision Matrix

	User Experience	Render Time	Optimization	Complexity	Maintenance
User Experience	1.00	5.00	6.00	9.00	9.00
Rendering Time	0.2	1.00	3.00	7.00	7.00
Optimization	0.17	0.33	1.00	3.00	3.00
Complexity	0.11	0.14	0.33	1.00	1.00
Maintenance	0.11	0.14	0.33	1.00	1.00

The next step that is necessary to have valid data is to normalize the matrix. To do so it is necessary to calculate the mean percentage of the Tables' 2 data. The mean is calculated as it is shown on Equation 1 and the results are showcased on Table 3.

$$\bar{x} = \frac{x1 + x2 + \cdots + x_n}{n} = \frac{1}{n}\sum_{i=1}^{n} xi$$

Equation 1 - Mean Equation

Table 3 - Normalized AHP Priority

Criteria	Priority
User Experience	58.7%
Rendering Time	23.2%
Optimization	9.9%
Complexity	4.1%
Maintenance	4.1%

Normalized data now is capable of providing accurate data analysis, and as such, the consistency ratio can be calculated.

Following:

$$CI = \frac{\gamma \max - n}{n-1} \; where \; n = order \; of \; the \; matrix$$

Equation 2 - Consistency Index

The calculation of the consistency ratio is possible following: $CR = CI/RI$

Equation 3 - Consistency Ratio

Where CI is calculated by following Equation 2 and RI is a random index to be used which has the value 10 in this scenario.

By applying Equation 2 to the values presented on Table 2, the consistency ratio has the value of 5.2%.

3.2. Information

After studying and researching the problem to be solved, it is important to determine the functions and uses of the product and its components. To do so, there are multiple techniques which can provide graphical representations of which scope the project has and how it is schedule, as well as the entire business model.

Project scope and schedule (roadmap/gantt) described in Figure 10:

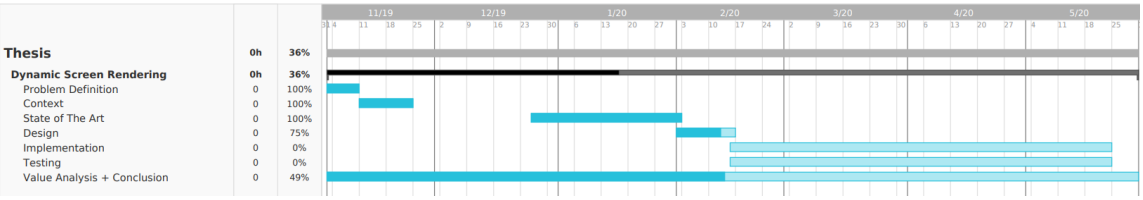

Figure 10 - Gantt Diagram

By analyzing the Gantt diagram, it is essential to understand the roadmap of how the project was planned how long each phase of the project takes. Figure 13 does provide a planning overview of how the Dynamic Screen Rendering problem was solved.

User and stakeholder needs' and values (CANVAS)

A CANVAS model is known for essentially having 9 different phases and it was proposed in [11] and it is a tool for managing strategic planning which allows visual representation of the existing or new Business Models. These 9 phases consist of:

- ***Key Activities***: Represent the activities/tasks that are needed to complete the business purpose of the product.
- ***Key Resources***: The resources that are needed to carry on the key activities in order to create value.
- ***Key Partners***: Having different partners to help carry on the key activities are important. These sections look to identify which partners are really necessary in order to accomplish the key activities. These partners can be either strategic, cooperation, joint ventures or buyer-supplier relationships.
- **Value Propositions:** Consists on the representation of the solution for a problem faced by a customer segment which creates value for it. It is the most important piece of the business model as the value is what will determine whether there is success or not within the business. As such, the value proposition should be innovative and disruptive within the market segment that the business is inserted on and it can be either quantitative or qualitative value.
- **Customer Segments**: It is the representation of the different segments of customers identified that are expected to meet the requirements of the value proposition of the business. This segmentation is done based on different geographical areas, age, gender, etc. This allows to have a better view of which market a business is segmented on and based on that adapt the approach to reach each customer segment.
- **Channels**: Represent the way that the value is brought to the customer and how the customer connects to company providing a product.

- **Customer Relationships**: This segment establishes which type of connections are hold between a company and the customers. The most common relationships are often:
    - Personal Assistance
    - Dedicated Personal Assistance
    - Self-service
    - Automated services
    - Communities
    - Co-creation
- **Cost Structure**: Identifies all the costs that are associated with the entire process of product creation and maintenance.
- **Revenue Streams**: The sources from which the company generates money by selling the product to customers.

The value proposition of this thesis is the development of a dynamic screen rendering engine whose screens are possible to be configured with relative ease without having to know technical details of the msg.Sales solution. As such, customers are able to generate screens themselves and customize them without having to require constant technical assistance provided from msg life Iberia support teams.

The Figure 11 represents the CANVAS model for the msg.Sales platform with the Dynamic Rendering Problem solved:

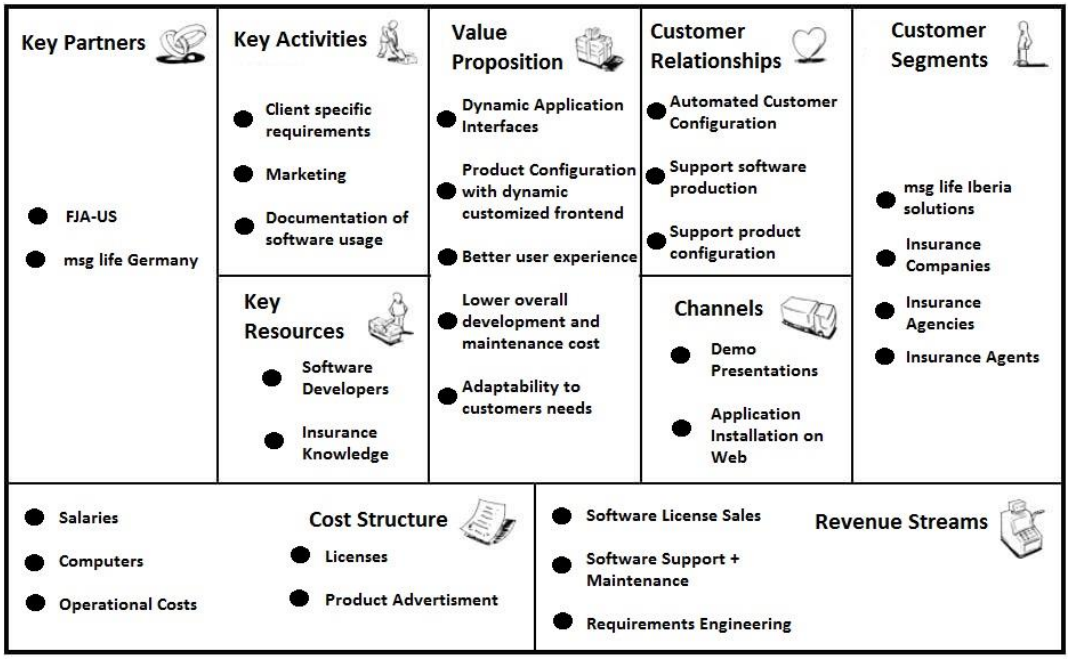

Figure 11 - msg.Sales Canvas

3.2.1. Innovation Process

The New Concept Development (NCD) model consists in the process of creating a new idea, concept or product into a market, where it is first necessary to come up with an idea followed by a design and market analysis phase. It has five influencing factors and they are represented on Figure 12:

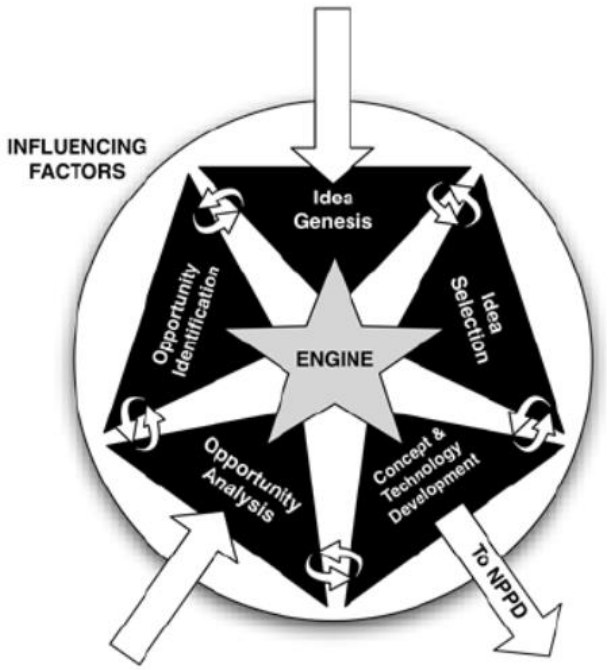

Figure 12 - NCD Model [12]

Each of these influencing factors have specific fields of domain, and as such, they have a set of methods and techniques that ease the analysis of them. These factors can be described as:

- Idea Genesis – Consists in the process of creating ideas that are built upon identified the identified opportunities. Brainstorming and idea banks are some of the techniques used on this stage.

- Idea Selection – The process of selecting an idea that resulted from the Idea Genesis phase. Companies do need to decide which idea they intend to pursue, and as such, evaluate and prioritize the ideas according to the planning, potential financial return and resource allocation.

- Concept & Technology Development – It is necessary to develop the idea that was identified within the company that strategically bring the most advantages. There are multiple techniques that can be applied [24]:

    - Goal Evaluation

    - Development planning that describes how the product can financially affect the company, the volume of the market and its growth rate.

- Opportunity Analysis – Segment that follows the Opportunity Identification. It is necessary to do a market analysis that fits within the company business strategy and it is done by doing analyzing and identifying the individual needs of insurance companies. This identification was previously done on Section 2.1 where it is notable analysis such as the client's needs, market segment evaluation and strategic placement.

- Opportunity Identification – This phase consists in the identification of opportunities that can be either technological changes or business opportunities. As mention in [13] , Koen identified tools such as future scenario mapping, problem-solving methods, fishbone diagrams, individual insights and ad hoc approaches. Msg life Iberia is capable of acquiring more customers by showcasing a solution capable of allowing customers on the insurance market to design their own screens and configure them, while the system is capable of dynamically rendering them.

3.2.2. Client Value

The term value is a subjective as it can have multiple meanings. As such, when value is approached it is essential to identify the meaning behind it as it usually can be one of the following:

- Value – This term can be described as how much something is worth within a field. It consists on the monetary, material or worth of a service describing what is considered good, bad or desirable within a specific field. Value is different for each person due to it being influenced by the ideas or beliefs that a person can have and how much it impacts each individual.

- Value for the customer – It is the perception of what a product or service is worth to a customer compared with the different alternatives. The term worth evaluates whether the customer feels the benefits and services over what he does pay, and as such, it is the evaluation of the benefits that are brought against the cost [14].

- Perceived value – This type of value consists in the notion of success of a product or service that is largely based on whether customers believe it can satisfy their needs [15].

The development of the solution that this thesis aims does provide value to two different entities, msg life Iberia and the customers which use msg life Iberia solutions.

msg life Iberia

One of the goals of the Dynamic Screen Render is to be flexible and dynamic enough to be integrated within every single application that msg life Iberia has developed or is currently developing. For msg life Iberia, the use of the Dynamic Screen Render on its applications would be noticeable from the moment of its integration. This way, msg life iberia does not only improve the technological stack that is used internally, but also improves the general frontend of their applications.

Dynamic Screen Renderer allows configuration through each specific use, and with this, msg life Iberia does improve on how expensive the maintenance of their current products is and is also able to showcase the improved software in order to attract new customers by showcasing the improved user experience and reduced screen time rendering within the solutions.

By bringing the configurability and adaptability that the solution provides, msg life Iberia is also able to reduce on the general costs of the human resources. This means that to problems which affect the frontend, developers would have a single source of truth to identify issues and new screens. Optimizing the way frontend is rendered brings msg life lower costs per feature on the average frontend development.

In order to achieve this, msg life Iberia will need to invest resources in the research and development of a capable solution to fit its needs, sacrificing time and money in order to invest on the development of such solution.

Clients

Dynamic Screen Render is able to provide value to the clients by mainly improving their user experience and reduce the time to go through a business process workflow from start to end and enable fragmented rendering within a screen.

The existing clients will be able to improve the efficiency of their employees that use solutions from msg life Iberia since the overall experience that is presented to them is improved.

Insurance business flows are complex and the overall time that is taken for an employee to complete the daily work is improved in multiple ways.

The application will be able to follow the employee's rhythm and adapt the application screens accordingly to the current step that is being executed by the employee.

Sometimes, agents are paid hourly by Insurance Employees. In these cases, the performance improvements within the applications will on the long term allow insurance companies to reduce their human resources cost and optimize their efficiency.

Since the solution is expected to have a good amount of configurability, it is also possible for insurance companies to configure themselves different new screens that will automatically work and be included on the application.

The acquisition of such a configurable and dynamic application that fits each customer needs' does need to come with an increased price. As such, the clients need to sacrifice some additional resources in order to compensate msg life Iberia for the development of such application.

3.2.3. Value Network

Value can be analyzed and built upon networks where a node represents an entity and the connections are the deliverables between entities. In order to build and analyze value, normally either a Value Network Analysis is done following the model of Verna Allee or Porter's value chain framework is used.

Transforming intangible assets into negotiable forms of value is proposed being a result from doing a value network analysis. Citing [16] :

> "*Value network analysis offers a way to model, analyse, evaluate, and improve the capability of a business to convert both tangible and intangible assets into other forms of negotiable value, and to realise greater value for itself. Underlying this approach is an understanding that intangible, but nonetheless strong and dynamic relationships, and the intangible assets that make up and have an impact on those relationships, are the foundation of any successful business endeavor. Indeed, the future success of a company or organisation as a whole depends on how efficiently a company can convert one form of value into another."*

In order to analyze the value of a network analysis, [16] defines the need to address three different questions:

Exchange Analysis – Address the chain of value exchange between a network, identifying whether the network exchanges both tangible and intangible value, and identifying not only bottlenecks but also the state of optimization.

Impact Analysis – Identify how value inputs bring value to each role within the network. During this process, it's a must to identify potential opportunities for value conversion and value realization opportunities.

Value Creation Analysis – Converting intangible assets into negotiable value. As such, it is necessary to look at how each role contributes and adds value to a network. This means that the value of a business can be measured as the value that the business itself provides in addition to the values that are inherent of the business network it is inserted on.

As an alternative method to analyze the value of a business which consists on a value chain. Citing [17] :

> *"The idea of the value chain is based on the process view of organizations, the idea of seeing a manufacturing (or service) organization as a system, made up of subsystems each with inputs, transformation processes and outputs. Inputs, transformation processes, and outputs involve the acquisition and consumption of resources – money, labour, materials, equipment, buildings, land,*

administration and management. How value chain activities are carried out determines costs and affects profits."

In [28], value chains are presented as decision support tools that is capable of diagnosing competitive advantage and finding ways to enhance a business's value.

The value chain analysis model is described in Figure 13:

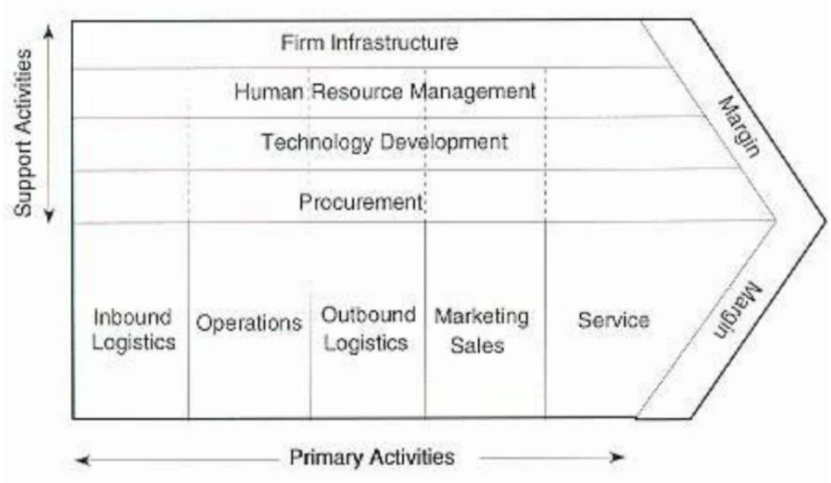

Figure 13 - Porters' Value Chain [28]

This model can be used to analyze the business value starting by identifying the sub activities for each primary and support activities. Afterwards, find the connections between all of the identified values in order to increase competitive advantage. Once all activities are identified, by reviewing each sub activity and link the last step is to enumerate the different possibilities that provide enhancement to the value that is offered to the customer

4. State of the Art

In this chapter it is described the major concepts that were used throughout the research process as well as techniques used in dynamic rendering.

4.1. Build Process

The build process of a software application is the process that transforms source code into software artifacts. A build process, is a well-defined set of procedures named pipeline that do normally consist of an alternative derived from the following sequence:

- Fetch most up to date code from a Source Control Repository
- Resources allocation and code compilation
- Run software tests
- As everything is going smoothly, build the software artifacts and store them
- Notify the build status

The build processed is defined in [18] as processes that are usually developed by build engineers who are specialized in optimizing build management systems.

There are various build tools which can be used to automate build processes such as:

- Make
- Ant
- Maven
- Gradle
- Grunt

Having an optimized build process is proved to enhance the work-hour efficiency of employees as it was studied on [19] while making use of an automation tool.

4.2. Model to Model Transformation

A model to model (M2M) transformation is the execution of a process that is capable of transforming a source model into a target model. These model transformations do make use of rules in order to identify exactly how and which components of a source model should be transformed to the target model. As such, model transformations are an essential constituent in Model-driven Engineering (MDE) [20].

A source model is a model which can only be navigated through, and the source model is the model that is written as a result. M2M transformations are a hybrid of declarative-imperative language, where the declarative part consists in matched rules in languages such as OCL, and the imperative part the actual rule execution.

Domain specific languages are usually accompanied by an object constraint language (OCL) to define the constraints within a metamodel. OCL is a pure specification language, which means that expressions are guaranteed without side effects. This means that evaluated expressions simply do return a value without changing the model in cause. Another definition for OCL is expressed in [21] as a language intended to facilitate specification of model properties in a comprehensible way.

The process of going through the transformations mention above do have a common factor. That factor is the use of Model Transformations which typically, do follow the concept described on Figure 14:

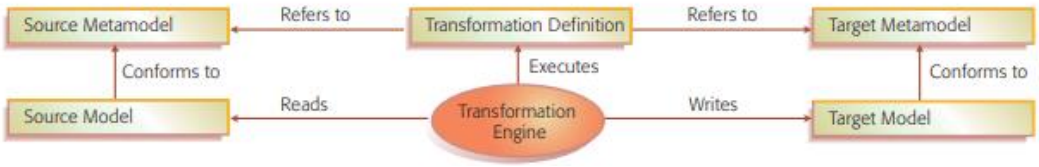

Figure 14- Model to Model transformation [22]

4.3. Web Performance Acceleration by Caching Rendering Results

This solution was developed and researched in [23] and it does make use of a caching mechanism in order to improve overall performance and user experience on the frontend of applications. In order to reduce the overall time taken to render a page on a web browser by using an in-network rendering function that renders web pages instead of web browser and caches the rendering results. After the initial caching is done, when re-rendering a page instead of interpreting the data again, the function reuses the results from the previous rendering and consequentially reducing the time taken to do each operation.

Overall, this kind of solution do not have any effect on the first rendering iteration of a web page, but does overall improve generically the rendering iterations of the same page with different content. Since a previous user might have rendered a similar page, by making use of the caching mechanism, this type of solution is able to make reuse the previous renderings and slightly adapt them to send the correct page to the web browser, as a result, displaying quickly the page.

Metadata can be completely different despite being the context of a rendered page being the same, and as such, the cache from a previous user rendering iteration can't be entirely used. To solve such problems, this solution proposes an approach displayed in Figure 15:

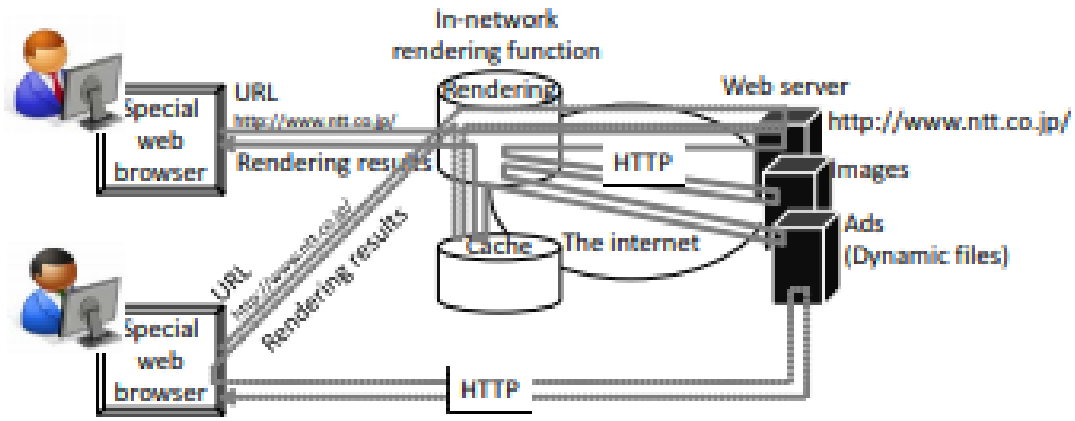

Figure 15 - Web caching architecture [23]

The main concept behind this solution is to determine static metadata that is recurrently needed to be rendered and reuse it. During the first iteration of a screen being rendered, the entire metadata structure is stored on a cache at the frontend level. The main goal at this phase is being able to identify which metadata from the entire screen is static for all consequent renderings. It is possible to do so by configuring the rendering function to store data of x amount of iterations and use that data as a model. To determine which data is considered static, for each screen rendering, a difference between the metadata between the current and the previous iteration is evaluated, removing the differences and caching the common factors while maintaining the DSL language that the rendering function knows such as scripts, images, JSON and XML. After x iterations, the cache know has the common metadata necessary for a specific page filtered, and when requested further renderings of the same page, the rendering function does no longer interpret the metadata entirely but makes use of the cache to provide the static content of a page. The dynamic content is applied by making use of an Extensible Stylesheet Language Transformation (xslt) and adding the dynamic content by applying xslt transformations.

The overall rendering performance is improved since after few renderings the static data will be more accurately determined and the rendering process will no longer have the need to interpret this data, but will directly make use of it.

4.4. Akamai Network

The Akami Network is one of the pioneers of content distribution networks (CDNs). In [24], a CDN is defined as the process of moving web content closer to clients by caching copies of web objects on thousands of servers worldwide.

In [25] an overall analysis was made for the entire systems of the Akamai Network and how it benefits applications in the Web.

The Akamai introduced Edge Computing services which are capable of holding deployments made by companies of their Java J2EE applications or specific components on the Akamai's edge. As stated in [25], Edge computing does benefit applications which have:

- ***Content aggregation/transformation***. These are relatively basic applications that do not require a transactional database. They simply collect content from Web services or other sources and reformat them for display (eg, using XSLT).

- ***Static databases***. Product catalogs, store locators, site search, and product configurators are examples of applications that use fairly static databases and can be run entirely at the edge.

- ***Data collection***. Many applications requiring forms or other user input can be handled on the edge, with data batched and sent to the origin asynchronously. For example, with a polling application, edge servers could store data locally and send results back to an origin server (or to Akamai 's storage system) in a few, larger chunks rather than many individual requests. Data validation and other types of basic logic can be executed by the edge server, without origin server involvement. This approach of aggregating content can reduce origin server load by several orders of magnitude.

- ***Complex applications***. Even with applications that require real-time database transactions, running presentation layer components of the application on the edge can still offer performance benefits, as origin server communications can be streamlined to include only raw data rather than full HTML pages. For example, the origin can generate a small dynamic page that references larger cacheable fragments, enabling the final HTML page to be assembled and served at the edge using Akamai 's ESI (Edge Side Includes) 9 technology.

As such, applications which do make use of dynamic data to render page are capable generating templates with static content which will be referenced when rendering a complex page with dynamic data, as static content that when all its smaller fragments are agglomerated will be capable of providing the final HTML pages while processing only small amounts of data (the dynamic content) and concatenating it within the HTML page by applying content transformation techniques such as XSLT transformations.

4.5. Concept Description Language

The use of a concept description language (CDL) to provide dynamic page rendering was studied in [26] and its main idea is to present the data in a semantic structure format, CDL, which does not depend on ontologies.

The architecture of the designed solution to provide dynamic content rendering while using CDL is shown in Figure 16:

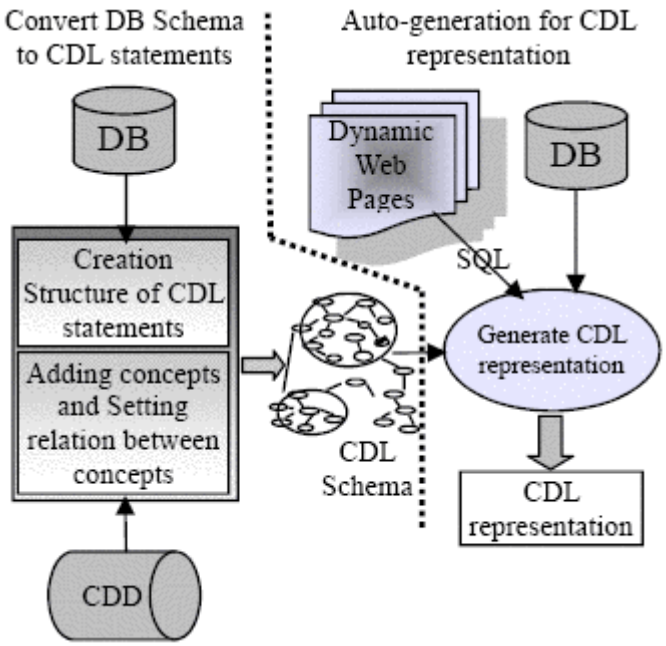

Figure 16 - CDL Architecture [26]

Database is where the metadata will be stored and as a result, the web page content will be generated by invoking REST/SOAP services which will be responsible for fetching the necessary metadata for a page by using SQL queries to a database. The generation of the CDL representation is done by applying semantic relationships between the metadata as their relations within the database such as foreign keys

represent them. Applying this process do generate a HTML template that are a representation of the CDL content that was applied to the fetched database metadata according to its schema.

It is possible to go further with this rendering technique and create multiple semantic syntaxes depending on the way that it is necessary to render content on a webpage. Generalizing and externalizing the files that are defining the semantic syntaxes provide a way of dynamically changing the within of same page with different metadata, by providing at runtime different outcomes since the semantic syntaxes are read every time a page is rendered. There is of course the downside of time consumption on the screen rendering as a result of fetching the semantic syntax files constantly and another side effect which is the increase complexity of the semantics to filter the content that already exists on the database schema but is not intended to be shown.

4.6. Automatic Translation from HTML documents to XML documents maintaining metadata

Just as there are techniques that do transform and translate HTML to XML, the reverse can be applied. A technique to transform HTML to XML was researched in [27] where it is applied the translation-based learning (TBL) paradigm. By creating a transformation-based translator which is capable of translating HTML documents with specific styles to XML files by applying semantic rules which convert a specific HTML document to an XML document which is done by applying XSL transformations.

These technique's template XSL transformations mainly start by stripping the HTML tags and assigning semantically the correspondent node tag in XML notation. Afterwards, formatting tags which do not have any relevance are stripped in order to reduce the file size and chunk its' content.

In case that the opposite is done, by applying the same logic and the same paradigm it is possible to convert XML to HTML. Converting the XML nodes and interpreting its attributes information in order to transform them in the correspondent HTML semantic can be done. The first step to do so, is to create a dictionary which does map an XML node to an HTML tag. By going through the XML file interpreting its' metadata, the nodes are transformed into HTML tags ordered and hierarchically structured according to the metadata provided from the XML file.

4.7. Model to Code

A Model to Code transformation consists in applying a mechanism which transverses the internal representation of a model and by applying a set of rules, transforms it into a text stream.

As explored in [28], the generation of templates can be achieved by applying transformations from a model to text. As such, transforming a Model that can be represented through a DSL to HTML is possible by creating templates which are capable of interpreting metadata that is provided from the DSL to small fragments of HTML pages that are described within defined templates. The main transformation does have a main template which does invoke the small fragmented templates in a specific order to create a structured HTML file.

There are multiple template-based generation tools such as Velocity. The source model can be restricted by applying declarative queries through the transverse of the model affecting as such, the resulting text stream.

4.8. Extensible Rendering Engine for XML and HTML

In [29] the possibilities of developing a rendering engine capable of interpreting metadata from XML and HTML was researched and consequentially developed.

This rendering engine loads the input file which is either a XML or an HTML file, that once interpreted and parsed to a canonical model, triggers a recursive process which iterates the canonical model and transforms it to the target file in the target format. This process starts by dispatching a parser that parses first the tag/node notations and then the children and attributes of each.

For XML parsing with the goal of generating HTML, it uses XSL to transform the model that results from parsing the input file to a canonical model into the target file. The XMl parser knows how to interpret metadata such as css styles, html tag notations and attributes of nodes and transform them in a model that contains all this relevant data. Once the metadata is synthetized it is necessary to render the content. To do so, the XSL transformations are executed followed by rules which know how to process the data included within the transformed canonical model and by following pre-defined XSL rules, render HTML templates that hold the initial metadata parsed to the target format while maintaining all of its characteristics such as styles.

As such, by creating a rendering engine that is capable of generating HTML templates from a DSL, rules can be configured accordingly to the context that the rendering engine is executing on. Making use of a rendering engine that knows how to part metadata defined in a DSL, and consequentially by applying XSL transformations with predefined

rules that are bound to the context where the rendering engine is inserted, it is logical that it is possible to achieve the expected results on maintaining the relevant metadata throughout the entire transformation process while not losing the goal of having an automated mechanism that is capable of generating templates according to different metadata sets through time.

4.9. Solution Proposal

After analyzing the state of the art, it is important to reflect upon it and evaluate how the studied approaches and techniques addresses the problem in hand.

As such, it is necessary to evaluate each topic approached in the state of the art and it is represented in tables 4, 5, 6 and 7.

Table 4 - Approach Evaluation

Approach	Advantages	Disadvantages	Restrictions
Web Performance Acceleration	• Allows the use of caching mechanism • Universal solution that can be applied to any PDS • Improves performance over time • Uses the native web browser cache	• Does not provide any mechanism to generate HTML templates • Does not provide a full fledge solution for the issued problem • The DSL that is known by the rendering function has to be maintained and is not configurable • Metadata still is required to be interpreted at the frontend side	• Can only be used at runtime and not included within an application build process • Does not allow the configurability of models and automatically provide screen generation for them

Table 5 - Approach Evaluation pt2

Approach	Advantages	Disadvantages	Restrictions
Akamai Network	• Communication can be done through small HTML fragments instead of entire pages • Allows HTML fragments caching • Possibility of changing HTML fragments dynamically with transformations	• Does not exactly solve the issued problem	• Can only be used within the web and for environments on the cloud
Concept Description Language	• Doesn't have ontologies dependencies • Only needs to interpret metadata once • Allows configurability and opens doors for clients to be capable of configuring screens	• Can't be included on the build process since it models database schema fetched through queries to html • SQL queries must be optimized to not include metadata that is not required for the HTML rendering process	• Requires semantic structure formats

Table 6 - Approach Evaluation pt3

Approach	Advantages	Disadvantages	Restrictions
HTML to XML automatic translation	• HTML to XML and vice versa transformation is possible to be implemented • Can be included within an application build process to transform a Flow file into HTML templates	• It is designed to be used at runtime and not during the build process • Requires manual optimization for the chunks of data transformation that are not necessary to not be transformed	• Researched solution is only tested for HTML to XML transformation • Requires a dictionary that knows the mappings between HTML and XML notation
Model to code	• Configurable Rules • Flow integration and configurability • Allows generation of HTML representing models defined in DSL's • Hierarchically transformations and maintaining order within HTML fragments • Build Process possible integration	• In order to be incorporated on the build stage of an application it is necessary to further develop as a standalone launcher to be invoked as a step of the build pipeline	• Requires the use of template generation technologies such as Velocity

Table 7 - Approach Evaluation pt4

Approach	Advantages	Disadvantages	Restrictions
XML and HTML extensible rendering engine	• Rendering engine capable of having as input XML or HTML files • Capable of having Flow files as input to define the screen being rendered • Use of XSL transformation provide lower time consumption • Allows to pass metadata such as CSS styles and attributes of xml nodes to the correspondent HTML notation • Automated page rendering mechanism capable of rendering pages configured by 3rd parties	• XML and HTML extensible rendering engine	• Rendering engine capable of having as input XML or HTML files • Capable of having Flow files as input to define the screen being rendered • Use of XSL transformation provide lower time consumption • Allows to pass metadata such as CSS styles and attributes of xml nodes to the correspondent HTML notation Automated page rendering mechanism capable of rendering pages configured by 3rd parties

By analyzing Tables 4, 5, 6 and 7 it possible to conclude that files and model transformations are predominant in the advantage's category. The most optimized solution would consist on being able to integrate the solution within the build process of an application, while being capable of transforming a Model to HTML files by applying rules defined on a DSL which is Flow files. As a bonus, while the application is deployed, apply caching mechanisms through iterations in order to determine the static metadata within the models that are sequentially used through page renderings.

As such, the approaches defined in section 4.2, 4.5, 4.6 and 4.7 are the ones which will be used as a base to design a solution adequate to fit within msg.Sales case.

The design of a solution capable of meeting all the goals of this thesis involves designing three different subsystems that do need to coexist in order to provide the full fledge solution. These three subsystems are:

- Dynamic Screen Renderer Engine
- HTML Render stage within a pipeline
- Cache mechanism for the HTML templates

The solution as whole can be described as a system capable of rendering HTML templates based on metadata provided from a PDS during the build process of the system, which can be used directly or indirectly during the process of rendering a screen which occurs within the Dynamic Screen Renderer Engine whose screens' configuration is made within Flow files. The consequent rendering of screens should be cached in order to improve the performance of the main system where this proposal is to be integrated within. A high-level architectural view is portrayed in Figure 17:

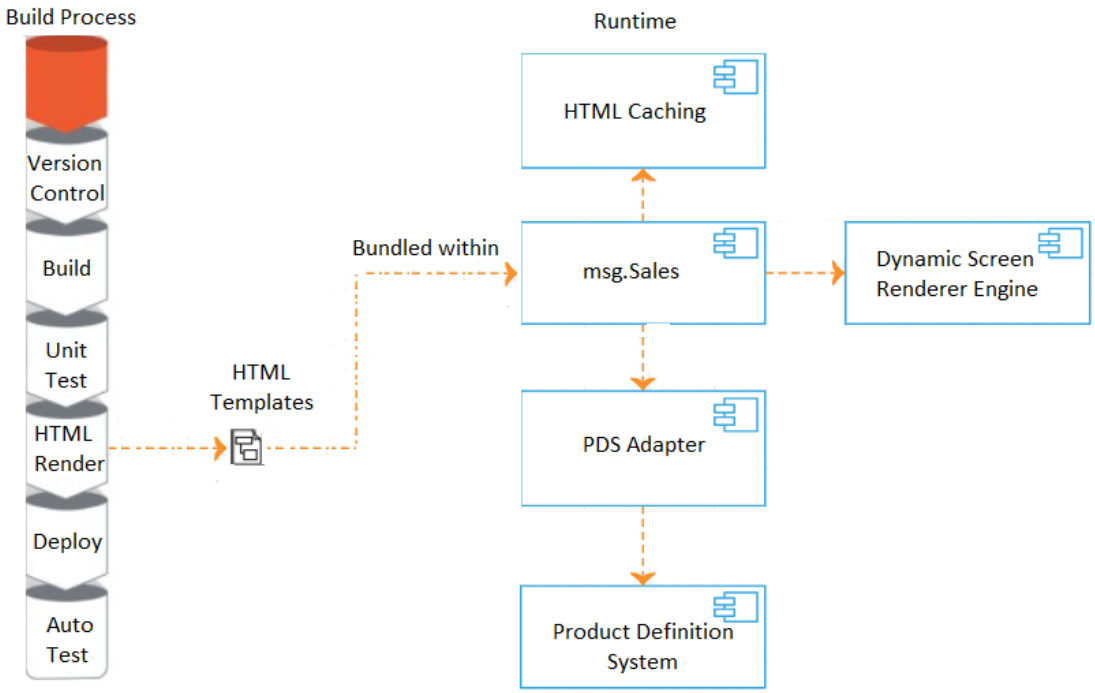

Figure 17 - Proposed Solution High Architectural View

The HTML template generator is included within the msg.Sales built artifact as a result of a specific stage within the build process pipeline. The flow files and their configuration are part of the msg.Sales component as well as the entire business layer. Every time that it is necessary for msg.Sales to obtain the most up to date product definition, he has to request it from the Product Definition System. To do so, he has to go through a

Product Definition Adapter which is responsible from transforming the internal BOM model from msg.Sales to the BOM of the current PDS implementation and vice-versa.

As soon as msg.Sales has an updated model, it is possible to render a screen to display it. To do so, msg.Sales interacts with the Dynamic Screen Renderer Engine and asks for it to provide a HTML file which is a representation of a screen that was previously configured via Flow files within msg.Sales. The Dynamic Screen Renderer Engine does receives context from msg.Sales (the flow file definition which represents what is intended to be shown on a screen and how it is going to be shown) that was previously transformed by applying the flow definition to the current msg.Sales model which ends up resulting in a new model that has all the metadata filtered and contextualized as it was configured on the flow files. Afterwards a Model to Code transformation is applied in order to transform this new model as to HTML pages.

Afterwards, msg.Sales is responsible for caching these rendered screens as they can be reused entirely if the end user for instance, tries to access the same screen within the same operation context.

The HTML template generator is intended to be used during the build process of the application, and as such, will only provide basic HTML pages that represent the initial screens of msg.Sales. This means that these templates will only hold information in regard to the basic structure of the model provided by the PDS initially, and will not contemplate any of its variances. This HTML generation will occur as soon as the unit test execution for a current build is marked as successful and before deploying msg.Sales to a specific environment.

The actual process of this stage within the pipeline is further detailed in Figure 18:

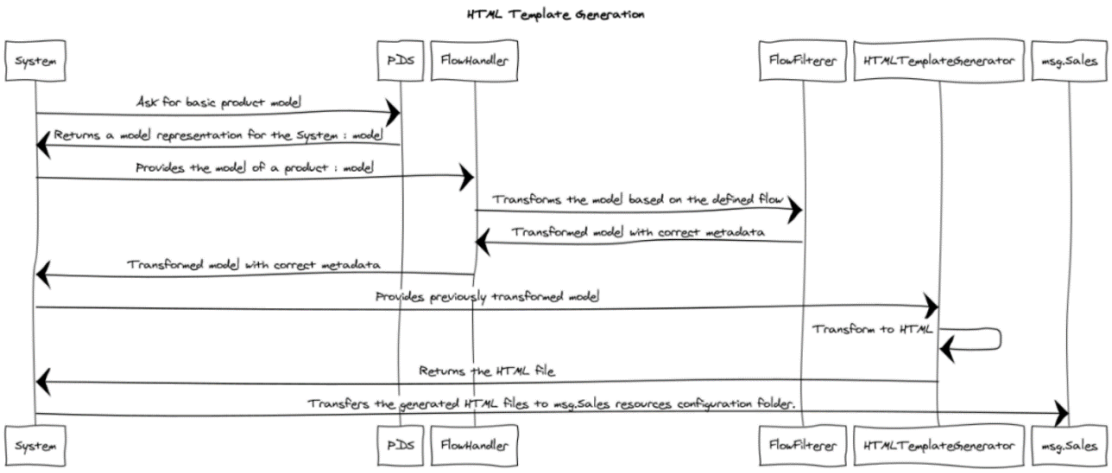

Figure 18 - HTML Template Generation Use Flow

This entire process is executed for each product that exists within msg.Sales, which means that it is necessary to obtain the product definition and consequentially render its HTML representation iteratively.

5. Solution Overview

In this chapter an extended analysis of the solution proposed in section 4.9 is thoroughly explained.

5.1. HTML Render on Build Pipeline

Since one of the goals for this thesis is to incorporate the solution within the build pipeline of msg.Sales in order to possibly improve the runtime performance, the solution has to be executable from outside of msg.Sales. As such, the pipeline stage that is responsible for handling this behavior is displayed in Figure 19:

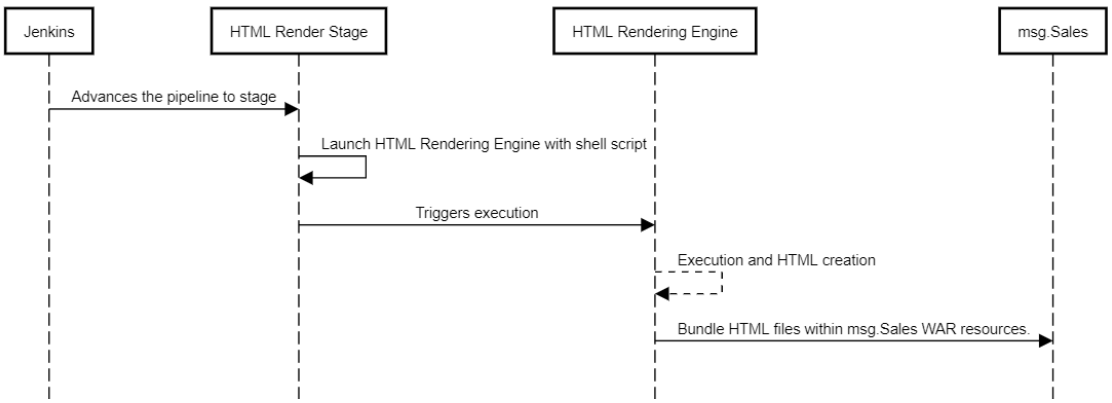

Figure 19 - HTML Render build pipeline stage

The behavior of HTML Rendering Engine is simplified and focused more on the build stage aspect as it will be further explained in 6.2, due to a more complex course of actions in regards to the HTML Rendering Engine integration with msg.Sales.

The sequence diagram showcased on Figure 19 can also be translated to a Jenkins Pipeline stage, as it is portrayed in Figure 20:

```
stage('HTML Render') {
    steps {
        echo "<---------------------------------------------- Executing HTML Template Rendering Stage! "---------------------------------------------->"
                        if(isUnix){
                                sh "java -jar ./HTML_Render/HtmlRenderer.jar"
                                cd /tmp
                                unzip ./MSG_Sales/target/msgSales-jboss.war
                                cp ./../HTML_Renderer/HTML_Files ./tmp/WEB-INF/classes
                                zip -r -u msgSales-jboss.war WEB-INF
                                mv -f msgSales-jobb.war ./../MSG_Sales/target
                        }else{
                                bat "java -jar ./HTML_Render/HtmlRenderer.jar"
                                cd /tmp
                                jar tvf ./../MSG_Sales/target/msgSales-jboss.war/ ./
                                copy ./../HTML_Renderer/HTML_Files ./tmp/WEB-INF/classes
                                jar uvf msgSales-jboss.war WEB-INF
                        }

    }
```

Figure 20 - HTML Render Stage Configuration

The main goal consists in adding the generated HTML Template files from the execution of the HTMLRenderer into the msg.Sales resources.

As Figure 20 shows, the operating system where Jenkins is installed is important since the commands to execute the necessary block of instructions to accomplish the stage goals is different. The native isUnix function is capable of identifying whether the current node is running on a UNIX or Windows system returning true or false.

Once the environment where the current node is operating if identified, the execution of the Dynamic Screen Render Engine is possible of being accomplished. As such, the next logical step is to run the jar file which is generated from building the Dynamic Screen Render Engine. The result of this step is the generation of a directory full of the HTML templates that are to be further included within the msg.Sales application.

To do so, it is necessary to explode the .war file built during the Pipeline Stage which builds the msg.Sales application, and change the resources within the war bundle to include the newly created HTML Files. This is accomplished by doing a copy of the directory which includes the files that were created by the execution of the Dynamic Screen Render Engine into the exploded WEB-INF folder of msg.Sales war file. Afterwards, the next step is to bundle msg.Sales war file again with its updated WEB-INF definition, so that the future deploy stage of msg.Sales will deploy the updated war file with the HTML templates embedded.

5.2. Dynamic Screen Render Engine

Dynamic Screen Render Engine is the core of the solution that is proposed as a capable and efficient to the problem at hands. The Engine itself has to be able to convert the Flow files that are configured for the application into its HTML representation and, as such, maintain the dynamism and configurability that flow provides to the end user.

In short, the Dynamic Screen Render Engine has to be able to recognize the BOM of an insurance product defined in a given PDS, and create an appropriate HTML file including all the relevant data from the product that is in the end, configured on a flow file. As a result of running the Engine for a given product, it is expected in the end to have a set of HTML files that do represent each of the Flow files that are configured within msg.Sales.

5.2.1. Obtaining the Insurance Product Definition

The first logical step for the Dynamic Screen Render Engine to execute is to obtain a list of all available insurance products at a given date. To do that, it is necessary to convert the internal BOM of a given application to the BOM of the PDS. To do so, it is necessary to provide a layer of abstraction between msg.Sales and the PDS interface, capable of handling entirely all the communication.

With this in mind, msg.Sales needs to make use of a layer responsible for such feat, and make use of two specific concepts being them:

- Transformers
- Builders.

A transformer is no less than an entity responsible for handling the entire BOM from msg.Sales and transform it entirely to the BOM of a given PDS. This is a necessary step to execute before invoking the product definition service within the interface provided by a given PDS so that the service request provides the BOM recognized by the PDS and as such is able to successfully interpret the metadata provided by msg.Sales.

This is an important step mainly when handling a BOM which was changed by interacting with msg.Sales application in order to keep data consistency. Since in this thesis the main goal is to handle the rendering of screens before the application is running on a web container, the transformation in this scenario is always going to be the same, meaning that the BOM which msg.Sales is going to transform before invoking the service is always going to be a mock. This is not an issue due to the fact that the PDS recognizes that the product definition that was asked is an initial definition without any changes.

After a successful invocation of the product definition service, depending on which PDS is configured to be used for msg.Sales, it is necessary to update the internal BOM with the product definition that was provided by the PDS. As such, the builders' layer exists to handle this specific situation and is responsible for handling the response of the product definition service by interpreting the returned data structure and building msg.Sales BOM with the new updated definition and its' values.

The Figure 21 showcases how the process defined above does work and how the interactions between each system is accomplished:

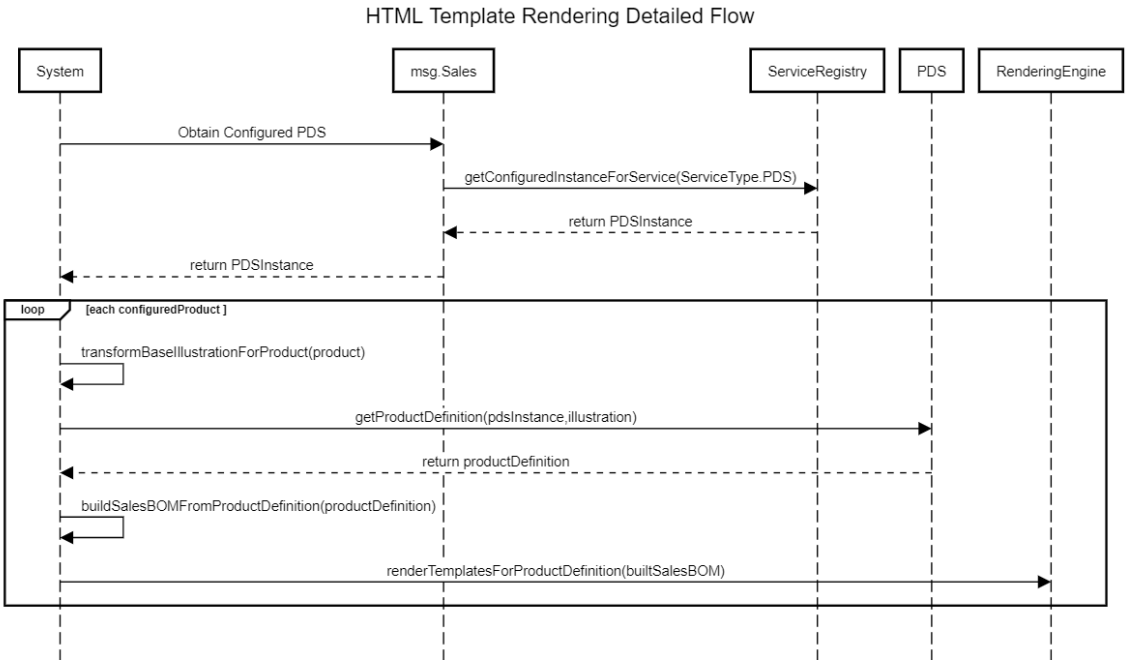

Figure 21 - Product Definition parsing process

This process starts with msg.Sales fetching from the **serviceRegistry** component the current PDS instance configured to be used. The **serviceRegistry** is a component that is configured using the XML notation and is responsible for having the configurations of all the services that are intended to be integrated with msg.Sales. It does also allow the configuration of multiple service instances for each service, which in the scope of this thesis, translates into having multiple instances for the PDS service. A sample of a configuration can be seen in Figure 22:

```
<?xml version="1.0" encoding="UTF-8"?>
<serviceRegistryConfiguration
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.msglifeib.com/serviceRegistry.xsd">
    <!-- Product definition service -->
    <Service type="PDS">
        <Implementations>
            <Implementation id="com.msglifeib.sss.adapter.lifefactory.pds.LifeFactoryPDSAdapter"
            configId="com.msglifeib.sss.adapter.lifefactory.pds.LifeFactoryPDSAdapter"
            serviceInstance="lfPDS" />
                url="/ipl-customer-lc-sc-server-webservices/service/lifeContractTransactionalServiceWS">
            </Implementation>
        </Implementations>
    </Service>

    <serviceInstances>
        <serviceInstance id="lfPDS">
            <baseUrl>https://baseurl:port/service</baseUrl>
            <serviceParams>
                <param name="authToken" value="insertTokenHere" />
            </serviceParams>
        </serviceInstance>
    </serviceInstances>
</serviceRegistryConfiguration>
```

Figure 22- Service Registry Sample

As the instance is successfully fetched, *msg.Sales* has now the necessary knowledge to establish a communication connection with the fetched *PDS* instance. In order to

communicate with the *PDS* as described in Figure 21, it is necessary to build a base Illustration context to provide as context for the *PDS*. This Illustration usually has a Contract Business Object attached which has an attribute that defines the productType. By transforming the Business Object from the system to the *BOM* supported by the *PDS*, the communication is now possible to be established, and as a result, the *PDS* returns to the System an updated representation of the entire *BOM* structure of an insurance product. Since the System does not recognize this returned *BOM* structured internally, it is necessary to build an internal *BOM* structure using the information given by the *PDS*, and as such, the builders' layer is as a necessary step to fulfill this need.

The next logical step is to be able to integrate the System with the Rendering Engine so that each insurance product definition can be translated into screens on the msg.Sales platform. To do so, the System has to be able to make use of the flow files that are configured to be used in msg.Sales and filter the product definition data to create appropriate screen templates for each configured flow file. This process can be described in Figure 23:

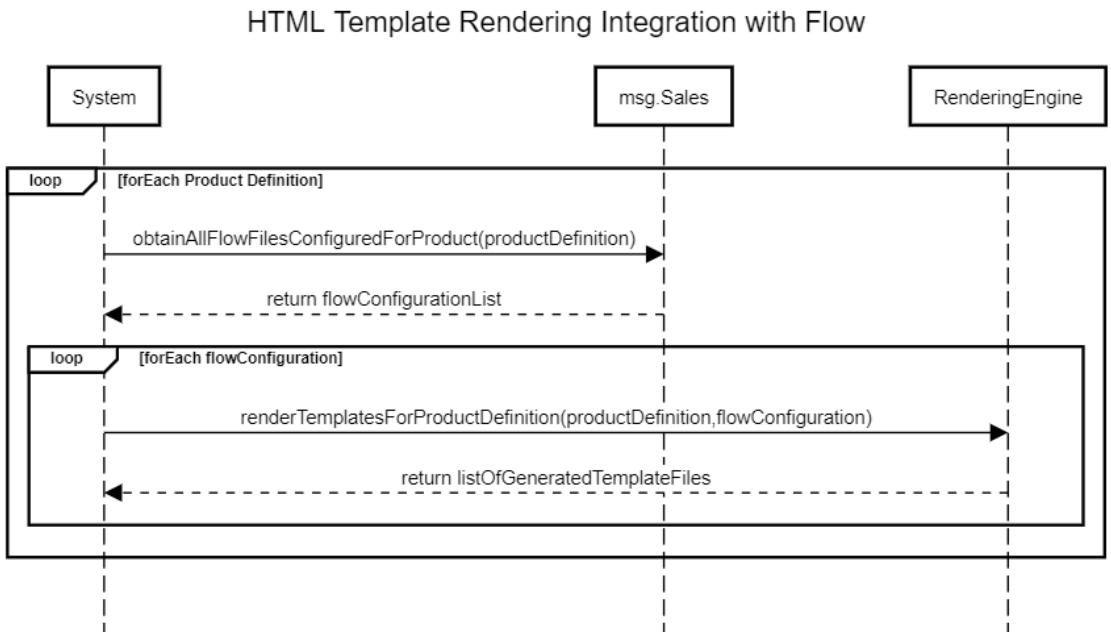

Figure 23 - HTML Template Rendering Integration with Flow

The first flow of actions described on the second loop of Figure 23 is a set of complex actions which do make use of the msg.Sales BOM in order to generate a HTML page. This complex process can be divided in two smaller steps which are:

- Process the BOM Instance according to a specific Flow file.
- Process BOM Instance + Flow metadata and process metadata to transform in HTML file.

5.2.2. Integration between Flow and Dynamic Screen Render Engine

To provide a summarized and incisive explanation on how this Flow and the Dynamic Screen Render Engine are integrated, it is first necessary to understand what exactly the Flow configured is here used for. The goal of a flow file is to describe how a screen within msg.Sales should be rendered and which metadata is relevant to a particular screen. This means that despite msg.Sales at a given moment in time having a complete and update BOM updated instance, a particular screen might only need half of the BOM metadata to accomplish its' goal. As such, the flow file describes the following:

- Which Business Objects are relevant to a screen within msg.Sales.
- Which metadata is intended to be processed within each Business Object.
- How is metadata intended to be shown within the msg.Sales screen.
- When a set of metadata should be shown within a screen.

Figure 24 does showcase an example of a flow file that does portray the topics mentioned above in a more detailed perspective:

```
<?xml version="1.0" encoding="UTF-8"?>
<Container xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.cor.fja.com/sss/xsd/Flow.xsd"
    defaultTriggerValidation="false">
    <BusinessObjects>
        <BusinessObject name="CustomerBO">
            <BusinessAttribute name="firstName" renderHint="TEXTBOX"
                required="false" relevant="true" viewMode="OUTPUT" sectionHint="customer-detail-row-1" />
            <BusinessAttribute name="secondName" renderHint="TEXTBOX"
                required="false" relevant="true" viewMode="OUTPUT" sectionHint="customer-detail-row-1" />
            <BusinessAttribute name="dateOfBirth" renderHint="DATETYPE"
                required="false" relevant="true" viewMode="OUTPUT" sectionHint="customer-detail-row-2" />
            <BusinessAttribute name="gender" renderHint="TEXTBOX"
                required="false" relevant="true" viewMode="OUTPUT" sectionHint="customer-detail-row-2" />
            <BusinessAttribute name="postalCode" renderHint="TEXTBOX"
                required="false" relevant="true" viewMode="OUTPUT" sectionHint="customer-detail-row-2" />
            <BusinessAttribute name="city" renderHint="TEXTBOX"
                required="false" relevant="true" viewMode="OUTPUT" sectionHint="customer-detail-row-3" />
            <BusinessAttribute name="street" renderHint="TEXTBOX"
                required="false" relevant="true" viewMode="OUTPUT" sectionHint="customer-detail-row-3" />
            <BusinessAttribute name="houseNumber" renderHint="TEXTBOX"
                required="false" relevant="true" viewMode="OUTPUT" sectionHint="customer-detail-row-3" />
            <Instance name="HasChildren">
                <InstanceCriteria>
                    <InstanceCriterion field="numberOfChildren"
                        valueNotEqual="0" />
                </InstanceCriteria>
                <BusinessObject name="CustomerBO">
                    <BusinessAttribute name="firstName" renderHint="TEXTBOX"
                        required="false" relevant="true" viewMode="OUTPUT" sectionHint="children-detail-row-1" />
                    <BusinessAttribute name="secondName" renderHint="TEXTBOX"
                        required="false" relevant="true" viewMode="OUTPUT" sectionHint="children-detail-row-1" />
                    <BusinessAttribute name="livingWithParents" renderHint="CHECKBOX"
                        required="true" relevant="true" viewMode="OUTPUT" sectionHint="children-detail-row-1" />
                </BusinessObject>
            </Instance>
        </BusinessObject>
    </BusinessObjects>
    <Links>
        <Link name="customer_back-to-search" labelId="Back" actionRight="partner.search"
            title="Go Back to Search" rel="customer-back-to-search" sectionHint="customer-detail-buttons-section" />
        <Link name="customer_select-customer" labelId="Select Customer"
            actionRight="illustration.partner.select" title="Select Customer"
            rel="customer-select" sectionHint="customer-detail-buttons-section" />
    </Links>
    <LayoutHints>
        <LayoutHint name="customerManager" />
    </LayoutHints>
</Container>
```

Figure 24 - Flow File Sample Configuration

This flow file configuration is a possible configuration for a screen within the msg.Sales, more specifically a screen where the details of a Customer are shown. There are some important aspects and notation here that is first required to be understood:

- Container – Main Complex Type which holds all the relevant data
- BusinessObjects – Complex Type which has a set of children BusinessObject
- BusinessObject – Complex Type which holds the metadata intended to be shown on a screen that is a specific Business Object
- BusinessAttribute – Simple Type that has refers to a specific block of metadata and defines how and where this metadata should be rendered
- Instance – Capable of conditionally show metadata depending on an Instance Criteria
- Instance Criteria – Defines all the necessary conditions to be fulfilled

- Instance Criterion – Child element of an Instance Criteria which is a condition himself
- Links – Container of Links that are intended to be shown on a screen (for instance buttons)
- Link – Single element that does provide the metadata of a button within a particular screen.

This flow file in the end is defining a screen within msg.Sales that is going to be shown when within the application a user does try to access the details of a specific customer. The flow file is determining that metadata relevant to a Customer will only be shown when the msg.Sales BOM does have a Customer Business Object, and the information which will be showcase on the screen is what is defined within the Business Attributes. Buttons which have custom behavior within a screen are also defined in the flow file in the Links section.

With the information mention above in mind, in order to render an HTML template for the screen described in the flow file described in Figure 24, it's necessary to process the information of the updated product definition provided from the PDS at the same time that the flow file is being processed which is the process described in Figure 23.

The next logical step is to generation of HTML templates by simultaneously processing the metadata from the PDS and the Flow configurations for each product.

5.2.3. HTML Template Generation

This is the most technical complex stage of the solution as whole, as it does process a huge amount of dynamic metadata. To establish a relation between the Flow configuration and the msg.Sales BOM for template rendering, the first step is to be able to map the Business Objects defined in the Flow to a Business Object within msg.Sales BOM. This is necessary since the parsing of the metadata within Flow and msg.Sales BOM will be done at the same time, and as such, will be used together as context for the Rendering Engine. An example of mapper configured used to achieve this is displayed in Figure 25:

```
<?xml version="1.0" encoding="utf-8"?>
<flowAndSalesMappings>
    <flowAndSalesMapping name="BusinessObjects">
        <entry key="SSSIllustrationBOAdaptable">IllustrationBO</entry>
        <entry key="SSSContractBOAdaptable">ContractBO</entry>
        <entry key="BankAccountBO">BankAccountBO</entry>
        <entry key="SSSCustomerBOAdaptable">CustomerBO</entry>
        <entry key="SSSPropertyBOAdaptable">PropertyBO</entry>
        <entry key="SSSPropertyGroupBOAdaptable">PropertyGroupBO</entry>
    </mapping>
</mappings>
```

Figure 25 - Simplified msg.Sales and Flow Business Object Mapper

The next step to generate a final HTML Template for a screen is to process the msg.Sales BOM instance which is updated and fetched from the PDS together with the Flow configurations for all screens that are configured for that product.

This process is illustrated in Figure 26 and will be further explored:

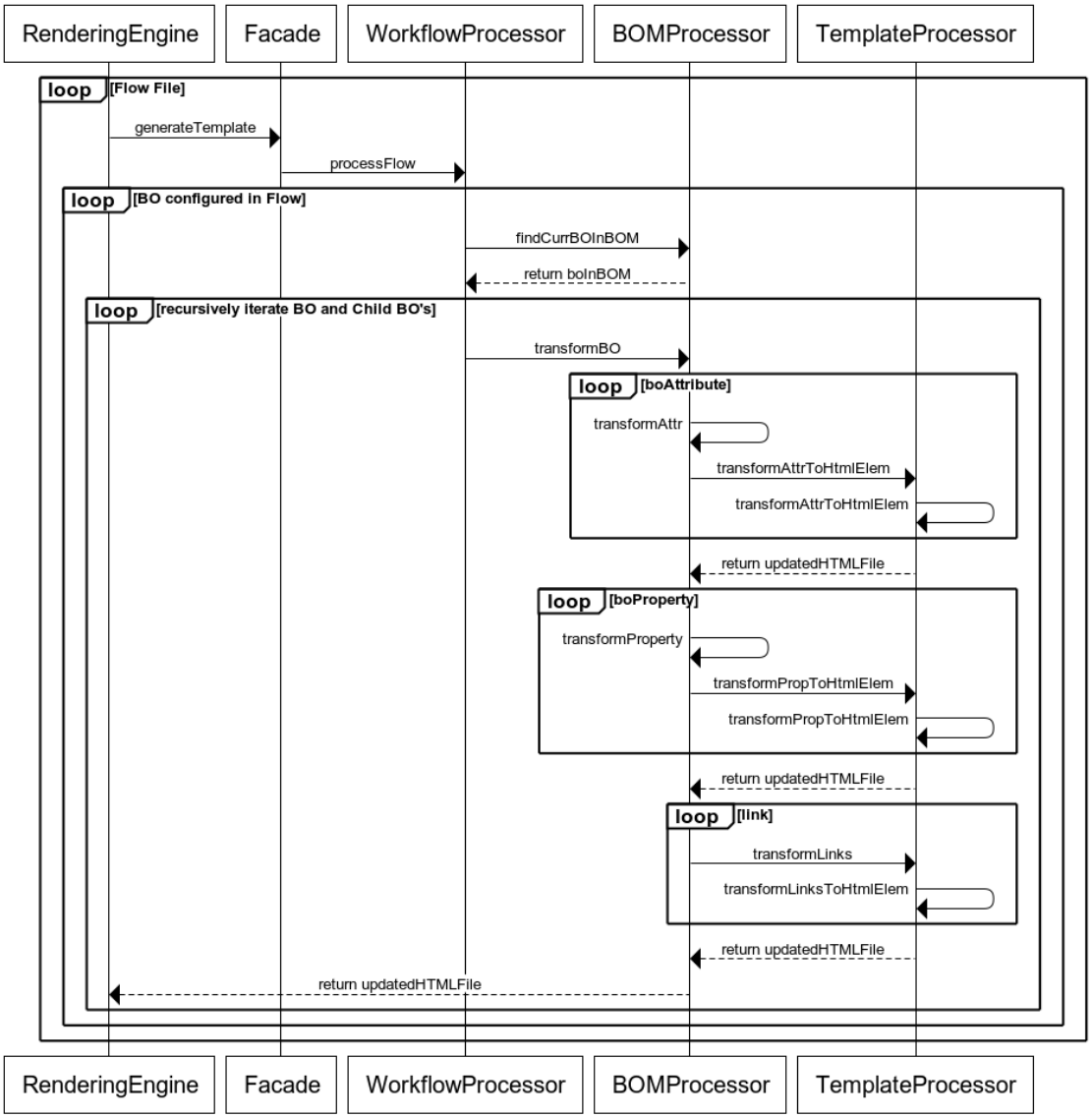

Figure 26 - Complete Rendering Engine Internal Processing

Since there are all the screens within msg.Sales application are configured from a Flow file, the goal is to generate a complete screen with the base product definition metadata filled for each existing Flow file. As such, the Rendering Engine has first to find all existing Flow configuration files, and start the HTML template rendering process for each Flow file.

Each Flow file as illustrated in Figure 24 has multiple BO's which themselves can have other children BO's or attributes. Since the Flow file determines which metadata is relevant for a screen, it is necessary to create a HTML Template by processing the BO hierarchy defined in the Flow. To do so, the Rendering Engine has to manipulate simultaneously the BOM context provided from the Flow and the BOM context provided from the PDS definition. By manipulating both at the same time and processing its' data, it is possible to generate a template screen with the correct metadata using the PDS definition whose metadata was filtered accordingly to the Flow configuration.

The process of transforming a Flow file consists in two different stages: processing the BOs' and processing the Links. Processing the BO's is no less than iterating the Business Object hierarchy defined in a Flow and processing each attribute/children BO's within the Flow. This involves not only processing the metadata within the Flow but as well the metadata within the PDS BOM definition. An example of this can be in Figure 25 sample, having another Business Attribute whose **renderHint** is a **select**. This results on the Rendering Engine using this element is to create an HTML select element whose domain values are fetched from the correspondent attribute in the PDS BOM.

The process of generating the HTML links is different in the sense that it does not require to have any kind of interaction with the PDS BOM. The links are not attached to the BOM in any kind of way, and as a result, they only requirement for them to being rendered on a page is to be configured on the Flow.

In order to achieve a coherent template rendering through multiple iterations, the rendering engine has the need make use of Velocity Template Engine in order to render the msg.Sales HTML page. Velocity Template Engine makes use of a file in the **.vm** extension which is no less than a template which is capable of being processed within Velocity by providing a context. This context allows msg.Sales Sections, Links and Business Objects to be passed as parameters to the Velocity Engine, being able as such to create an .**html** page capable of representing a screen accordingly to the msg.Sales BOM and processed Flow.

The main template is displayed in Figure 27, and it triggers the chaining of element intended to be on a particular screen:

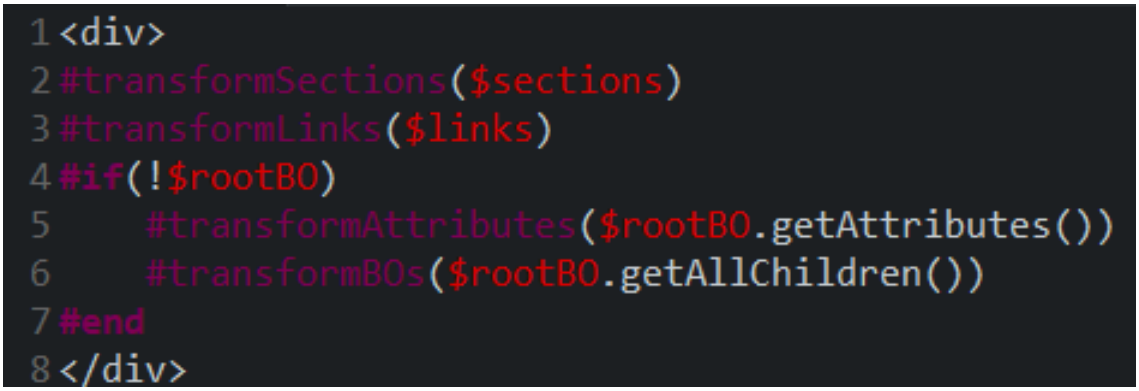

Figure 27 - Main Velocity Transformation Template

This template is looking to be able to create an HTML fragment which does represent the content of an msg.Sales screen. Variables are appended with a prefix **$** and the use of # provides context to the Velocity Engine that it is the equivalent of a macro execution. In this scenario, the goal is to create all the sections and links defined from the Flow to a screen, and afterwards in case that the Flow is properly configured and has a root object, render the HTML correlation of its attributes and child BOs. This means that the **#transformSections, #transformLinks, #transformAttributes** and **#transformBOs** triggers the execution of 4 smaller macros which are responsible for handling different type of content.

The **#transformSections** macro does need to have as context which were the sections are defined in the Flow, and as such, render the sections themselves as they are defined, using the template in Figure 28:

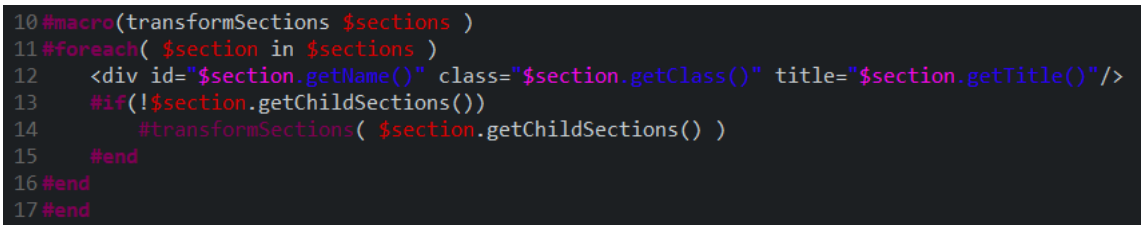

Figure 28 - Velocity Template Render Sections

Execution the **#transformLinks** is does transform the links defined in the Flow into contextualized HTML elements. This macro can be seen in Figure 29:

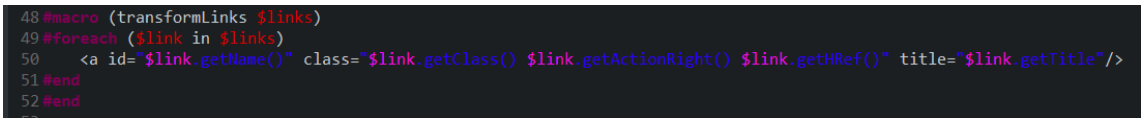

Figure 29 - Velocity Template Render Links

Each Business Object is capable of having multiple attributes, and in case that the Business Object is a **Property BO**, he is capable of having Business Properties configured within the Business Object definition. As such, it is necessary to have macros

configured that are capable of handling both of these scenarios, being them showcased in Figure 30.

.

```
#macro(transformAttributes $boAttributes)
#foreach( $boAttribute in $boAttributes)
  <$boAttribute.renderHintAsHTMLElement() class="$boAttribute.getVisibility()
  $boAttribute.getViewMode()" id="$boAttribute.getName()" />
#end
#end

#macro(transformProperties $boProperties)
#foreach( $boProperty in $boProperties)
  <$boProperty.renderHintAsHTMLElement() $boProperty.getRequired()  id="$boProperty.getTag()" />
#end
#end
```

Figure 30 - Velocity Template Render Attributes and Properties

All that is left is to group everything together in order for the code in Figure 27 to execute is to handle the processing of the BOM using recursion. This is achieved by the macro configured in Figure 31:

```
#macro(transformBOs $childBOs )
#foreach( $childBO in $childBOs)
    #transformAttributes($childBO.getAllAttributes())
    #if($childBO.isProperty())
        #transformProperties($childBO.getAllProperties())
    #end
    #if($childBO.getAllChildren().size() > 0)
        #foreach( $innerChildBO in $childBO.getAllChildren())
            #transformBOs($innerChildBO)
        #end
    #end
#end
#end
```

Figure 31 - Velocity Template Render BOM Recursion

This is the final step that is executed through the Dynamic Screen Render Engine as it results on a HTML fragment that when appended to the main msg.Sales HTML fragment, represents the screen that is intended to be shown exactly the same way as Flow describes it.

5.3. HTML Caching System

In order to optimize even further and reduce the time that each rendering action takes within the msg.Sales system, the implementation of an HTML Caching System goes along with the development of the Dynamic Screen Render Engine. Now that the screens on msg.Sales are no longer rendered fully on runtime, since the generated HTML resources according to all configured Flows will no longer change unless a new build of msg.Sales occurs, it is no longer necessary to continuously invoke services that return a representation of a screen since the screen was already previously generated. As such, after requesting a screen for the first time, it is enough for msg.Sales to cache all the screens that were previously rendered.

This means that the downtime between msg.Sales API and msg.Sales frontend interaction to render a screen is capable of being reduced since the HTML fragment itself is capable of being reused directly without needing the API to do any processing in regards to it.

Figure 32 portrays how this cache works in between the communication of the Browser and the msg.Sales API:

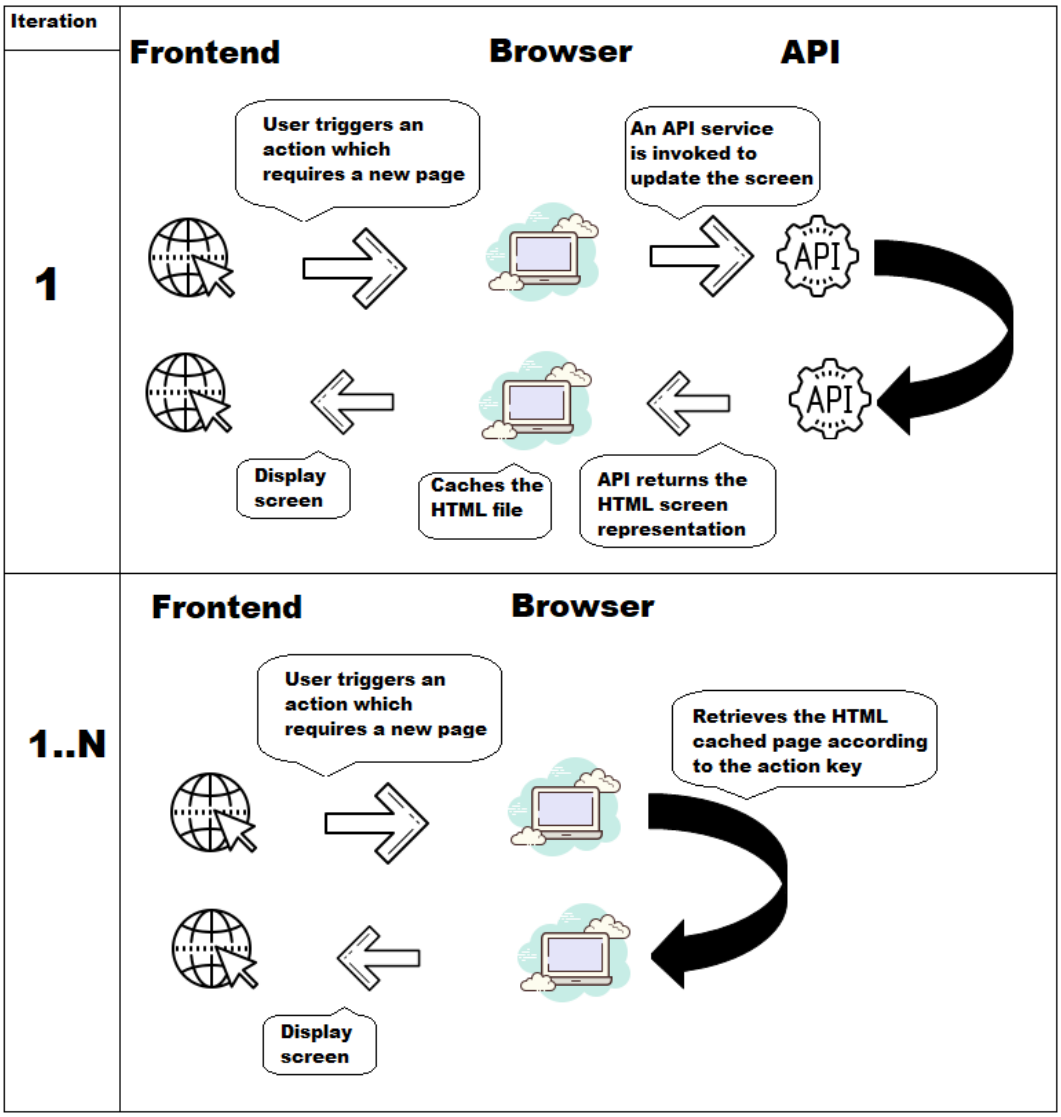

Figure 32 - HTML Browser Cache

The goal here is to cut the time that is consumed while invoking the API in order to obtain a particular screen, assuming that the screen definition will not change throughout time.

There are two different implementations of browser caching mechanisms which suits msg.Sales needs', being them:

- Local storage
- Session Storage

Of these two, using the Session Storage does provide a better solution in the sense that msg.Sales sharing the cache between different tabs and browser windows from the same origin is an optimal feature alongside the data not expiring even after a browser restart or even an OS reboot.

A customized caching implementation is necessary in order to map actions within the msg.Sales implementation with its' correspondent screen representation. This association should occur on the moment that the application's routing is triggered, and routes to a specific screen within msg.Sales. This way, the frontend of msg.Sales is responsible for knowing whether an API request is necessary to obtain a screen representation or to reuse a previously retrieved screen.

Caching a screen does provide an additional performance improvement in screen rendering within msg.Sales, as cutting the time consumed by frontend interacting with msg.Sales API is negated when there is a cache representation of that particular API action. Figure 33 is a representation of the different alternatives within the cache system:

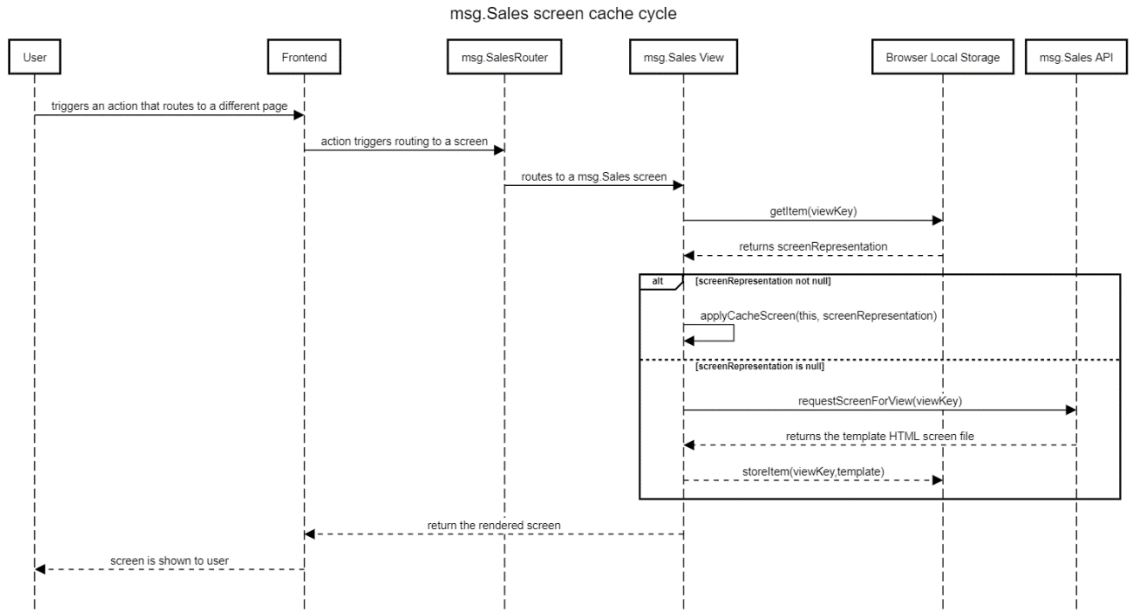

Figure 33 - Cache usage sequence diagram

6. Evaluation & Experimentation

The main goals of this chapter are:

- Definition of the alternative investigation hypothesis that allow the validation of the developed software
- Identify the different information indicators and correspondent sources.
- Describe how each information indicator and hypothesis is evaluated.

6.1. Hypothesis Definition

In order to evaluate whether the developed software met its' goals or not, it is necessary to define the different hypothesis which allow this evaluation to be done. As such, it is necessary to define the null hypothesis and the alternative hypothesis.

Null Hypothesis (H0):

- The developed software doesn't improve the rendering time of screens.
- The HTML template rendering mechanism based on the product definition system can't be integrated within the build process of an application.
- The developed software doesn't allow a user to dynamically define a screen that will be rendered.
- The developed software doesn't allow the implementation of an HTML templating cache.
- The developed software doesn't provide any improvement on the screen configurability process.

Alternative Hypothesis (H1):

- The developed software improves the rendering time of screens.
- The HTML template rendering mechanism based on the product definition system can be integrated within the build process of an application.
- The developed software allows a user to dynamically define a screen that will be rendered.
- The developed software allows the implementation of an HTML templating cache

6.2. Indicators and Information Sources

This section identifies which are the indicators and from which sources of information the evaluation of the defined hypothesis is made.

6.2.1. Indicators

Indicators are evaluation measurements that allow to confirm or reject the defined hypothesis. The indicators relevant to the evaluation of this thesis are:

- **Configurability –** It should evaluate the capacity that the software has in dynamic screen configuration that is intended to be rendered, allowing to define which information within msg.Sales canonical model should be propagated to the rendered screen. It should evaluate the different combinations that are possible for users to configure to reflect their own customized screen.

- **Integration –** It should not only evaluate the integration of a HTML template rendering system with the software build process but also the integration of a HTML caching mechanism within msg.Sales

- **Usability –** It should evaluate how the end user interacts with the system not only at the personal satisfaction level of software usage, but also at the ease of software usage.

- **Efficiency –** It should evaluate the capacity that the developed software provides in screen rendering time and configuration reduction.

- **Confiability –** It should evaluate the capacity that the software has in keeping the same level of performance and correct screen renderings in accord to their configurations while being fault-tolerant and capable of recovering the performance level even after failures.

- **Maintainability –** It should evaluate the capacity that the developed software has in being modified for improvements, extensions or bug fixing while being capable of keeping stable and avoid collateral effects resulting from software changes.

- **Portability –** It should evaluate the capacity that the developed software has in being integrated with other systems such as the different software that is developed by msg life Iberia.

6.2.2. Information Sources

In order to measure indicators, mention in Section 7.2.1, it is necessary to identify and use information sources. In the scope of this thesis, the following sources were identified as relevant:

- **Regression Tests –** Tests which are developed and applied to screens that were previously rendered before the development of the software, and evaluate whether they keep being correctly rendered or not after the integration of the newly developed software.
- **Stress Tests –** Tests and evaluates the confiability of the developed software by verifying the capacity that the software has in rendering multiple different screens with success at the same time.
- **Recovery Tests –** Tests the developed software by verifying that in case a failure occurs during the screen rendering process, the system is capable of using the HTML template that was rendered on the build process as the input to render the recovered screen.
- **Performance Tests –** Evaluates the time that a screen rendering iteration takes before and after the integration of the newly developed software to allow measurements which indicate performance improvements or degradations.

6.2.3. Evaluation Methodologies

In order to evaluate the hypothesis and verify which does apply, it is necessary for this thesis to follow a strict and well-defined evaluation process. This process follows the identified indicators and its sources of information to dictate truthful data.

Integration Testing – A set of integration tests are executed for the newly developed software, being them:

- Integration of the HTML caching mechanism on the other software developed by msg life Iberia
- Integration of the HTML template rendering mechanism within the build process of different software developed by msg life Iberia.
- Integration of the dynamic screen rendering system within the different software developed by msg life Iberia.

In order to evaluate the results for this kind of test, the application reports relative to a successful use of the three different subsystems is verified.

Regression Testing – The test batteries that msg life Iberia has for its different software should be applied after the integration of the newly developed software in order to evaluate whether the software keep meeting its functional and non-functional requirements successfully.

Stress Tests – A battery of automated tests that request the dynamic screen rendering API in extreme numbers is executed in order to evaluate its confiability. These tests should also include evaluations of software recovery to contemplate eventual failures evaluating not only the correct behavior of the system under stress, but also the eventual recovery for eventual possible failures.

Efficiency and Performance Testing – These tests will allow to obtain the rendering time of a screen before and after the integration of the software. This way, it is capable of evaluating the efficiency of the software and still evaluate the capacity that the software has in reducing the necessary time to configure a screen. These tests should also make use of a Control/Test group in order to contemplate different screen rendering possibilities and keep the coherence of them before and after the integration of the newly developed software. The efficiency should be evaluated by applying analysis of variance (ANOVA) tests since the input parameters will be the different metamodels which are capable of being individually evaluated. This way, the ANOVA test technique should be applied in order to obtain quantitative data measurements for the before and after the software integration.

6.3. Dynamic Screen Render Evaluation

As defined in section 9.1, the H1 hypothesis defines the main goals that are intended to be achieved with the implementation of the Dynamic Screen Render. The HTML caching mechanism will be evaluated independently in section 9.3.1, and as a full fledge solution in 9.3.2.

To use a base metric to determine the performance improvements given by Dynamic Screen Render and HTML Template caching, the time taken to render a screen within the msg.Sales system is the relevant comparison metric. The statistical performance test is scoped within three different msg.Sales screens which use different metamodels:

- Illustration screen
- Plan screen
- Documents screen

In order to correctly calculate the precision of the data samples, it is necessary to calculate the standard deviation and standard error for the data samples where the solutions are applied, by applying to each sample an ANOVA test. Figures 34, 35 and 36 represent respectively the original time taken to render each of the screens mentioned above:

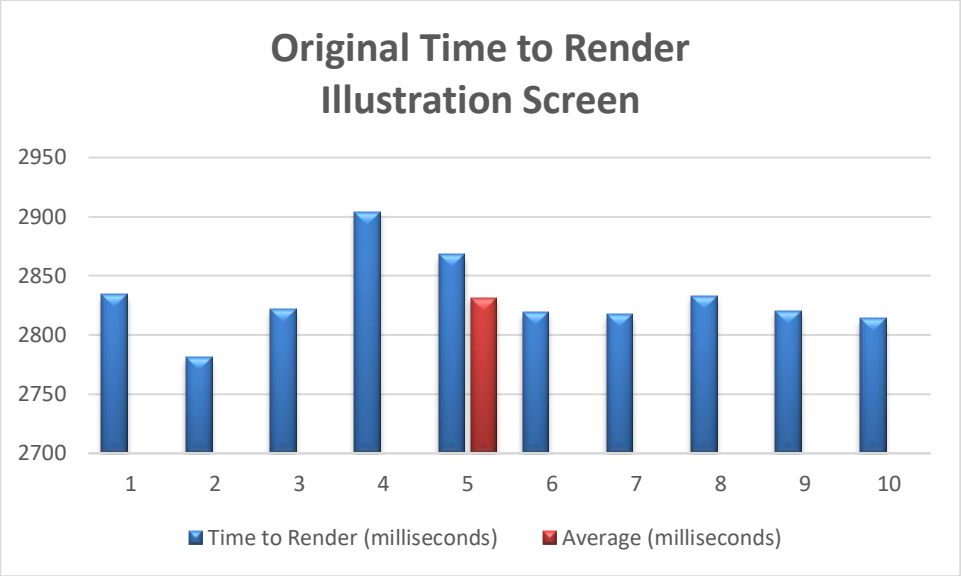

Figure 34 - Original Time to Render Illustration Screen

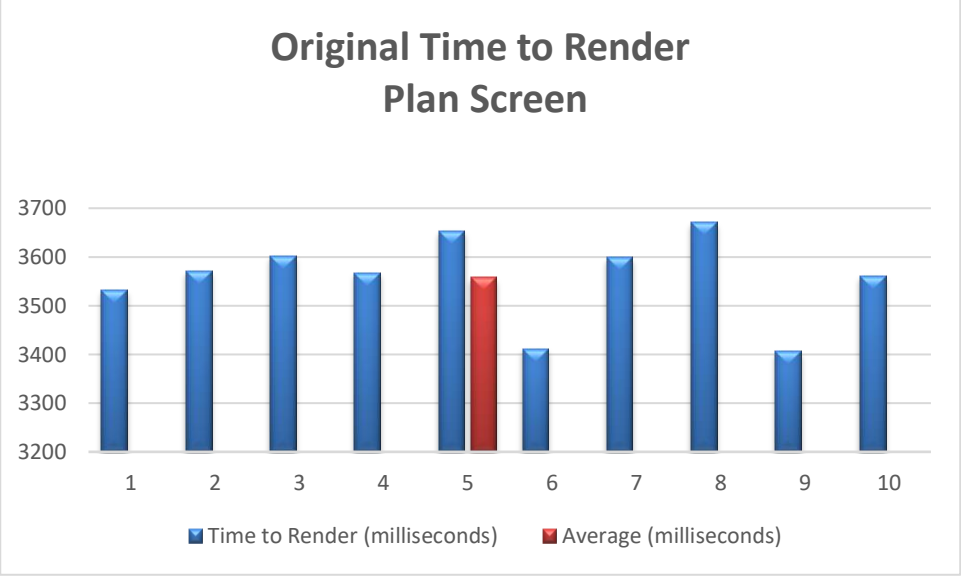

Figure 35 - Original Time to Render Plan Screen

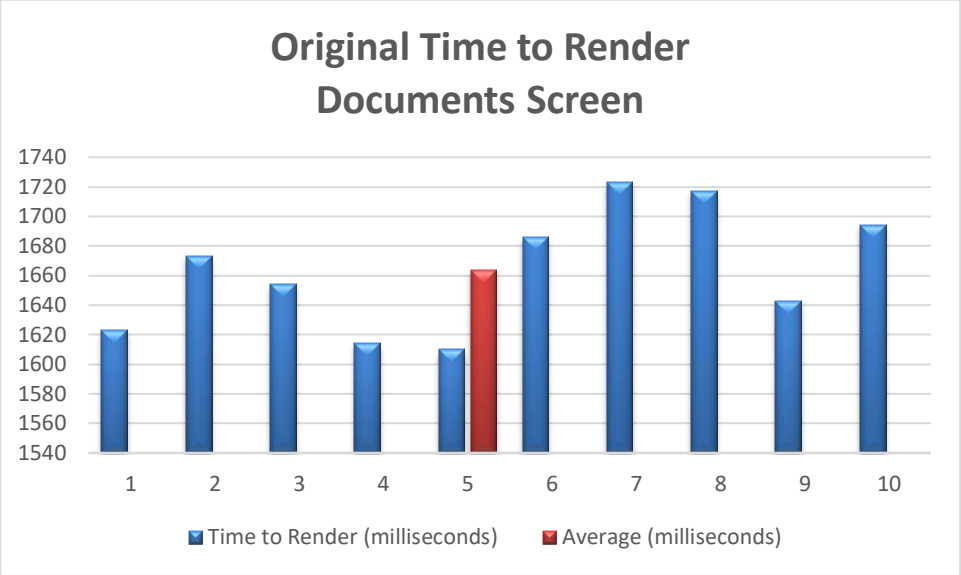

Figure 36 - Original Time to Render Documents Screen

To perform an ANOVA test between two samples, it is necessary two have two data samples. To do so, the data from Figures 37, 38 and 39 will be used to perform an ANOVA test.

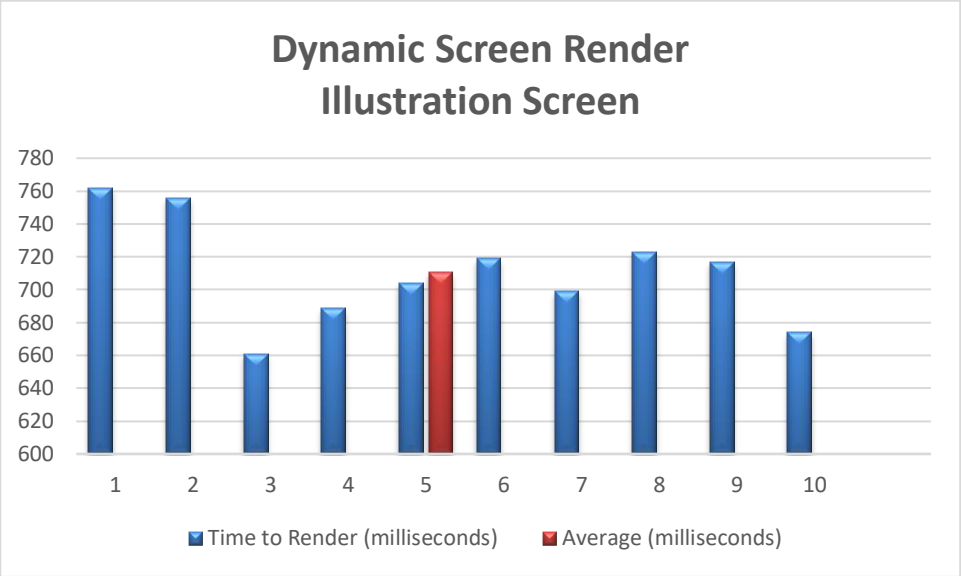

Figure 37- Dynamic Screen Render Illustration Screen

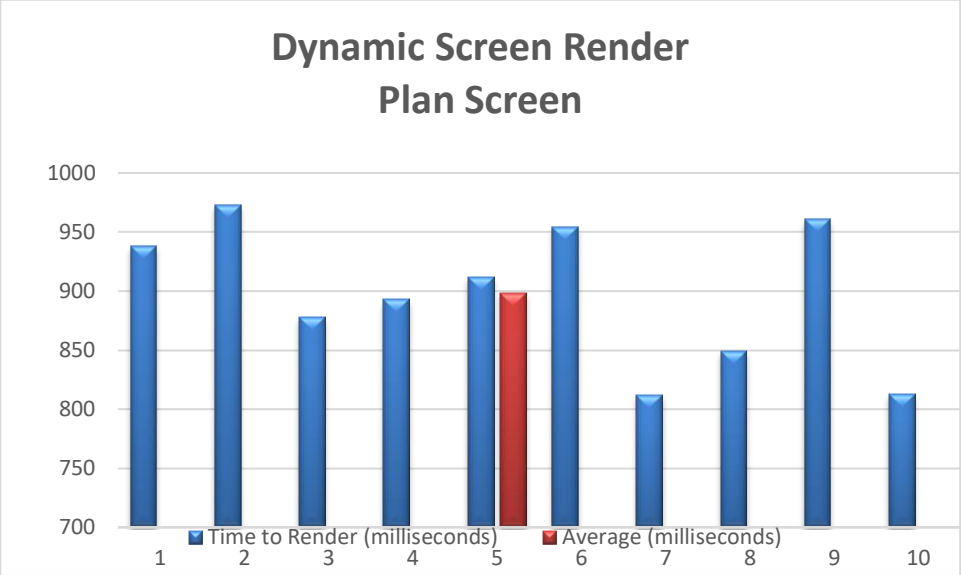

Figure 38 - Dynamic Screen Render Plan Screen

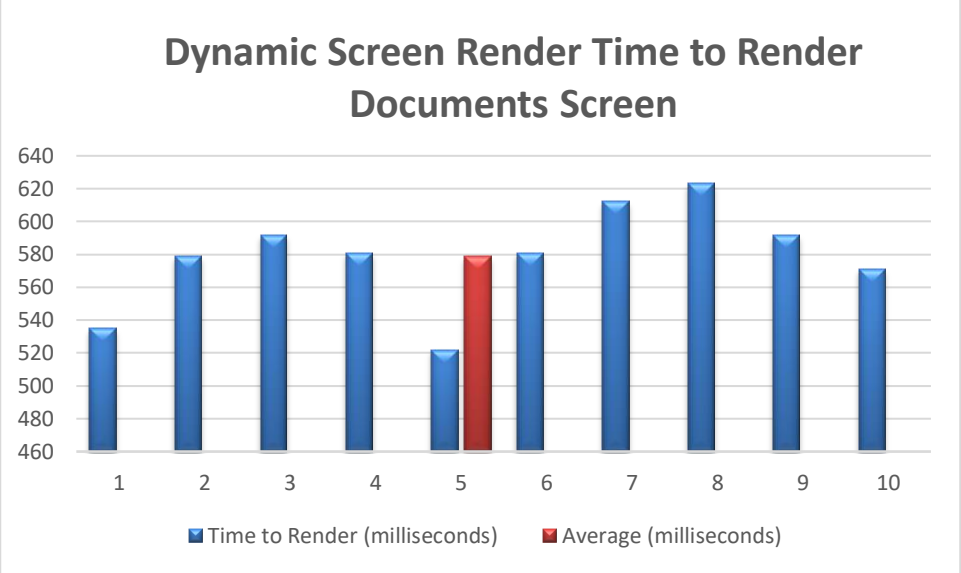

Figure 39 - Dynamic Screen Render Time to Render Documents Screen

Performing an ANOVA test between these two data samples on each of the screens results in Table 8, 9 and 10 correspondently:

Table 8 - ANOVA test between Original and Dynamic Screen Render on Illustration screen.

Groups	N	Mean	Standard Deviation	Standard Error
Original	10	2831.2	33.4026	10.5628
Dynamic Screen Render	10	710.4	32.3392	10.2265

Table 9 - ANOVA test between Original and Dynamic Screen Render on Plan screen

Groups	N	Mean	Standard Deviation	Standard Error
Original	10	3557.7	88.9457	28.1271
Dynamic Screen Render	10	898.3	59.6025	18.848

Table 10 - ANOVA test between Original and Dynamic Screen Render on Documents screen

Groups	N	Mean	Standard Deviation	Standard Error
Original	10	1663.7	41.355	13.0776
Dynamic Screen Render	10	578.8	30.9473	9.7864

Analyzing the data above it's possible to conclude that on average, the improvements percentage wise for each of the screens are as follow:

- Illustration screen – 74,93% improvement
- Plan screen – 74,75% improvement
- Documents screen – 65,5% improvement

According to the Standard Deviation and Standard Error values, with the implementation of the Dynamic Screen Render the time taken to render is not only reduced, but does also showcase the lower disparity of difference between iterations while not being volatile measurements.

6.3.1. Full Fledge Solution Evaluation

The evaluation of the caching mechanism should not be evaluated independently since it does only apply to consequent renderings of the same screen. This means that the first rendering should make use of the dynamic screen renderer while all the consequent renderings will make use of the cache as well. The base metrics considered for performing an ANOVA test in the Full Fledge solution which includes not only the Dynamic Screen Render, but also the HTML Template cache are represented in Figures 40, 41 and 42 according to each considered screen:

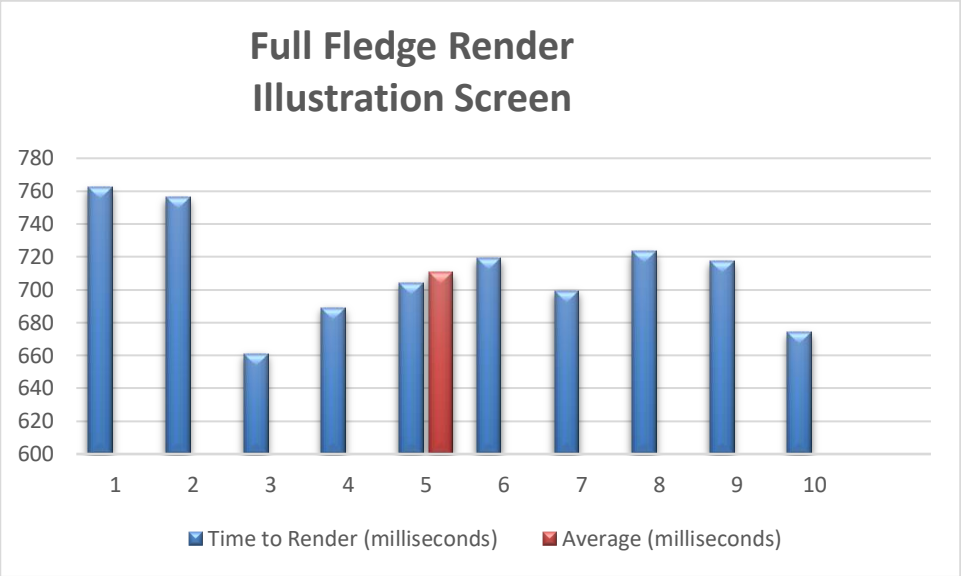

Figure 40 - Full Fledge Time to render an illustration screen

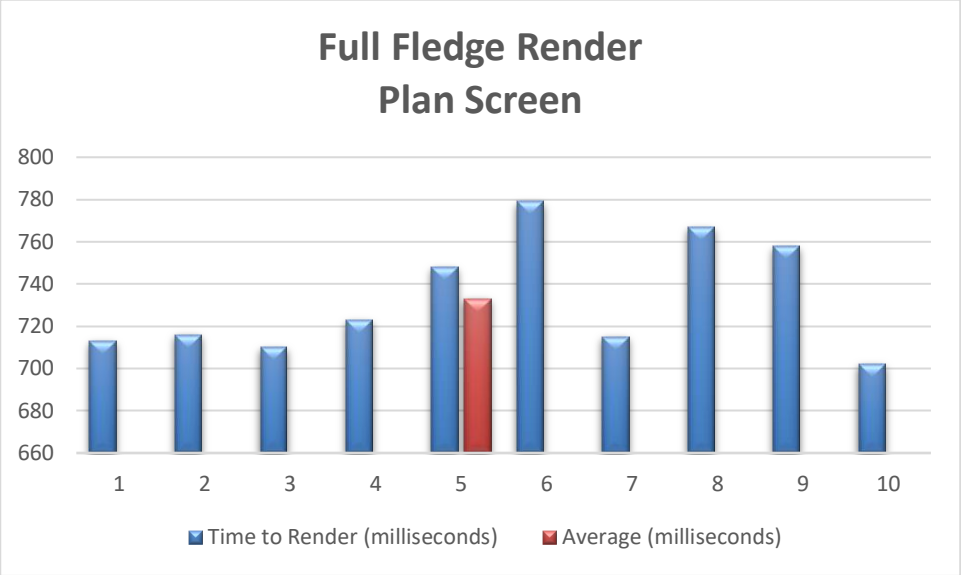

Figure 41 - Full Fledge time to render plan screen

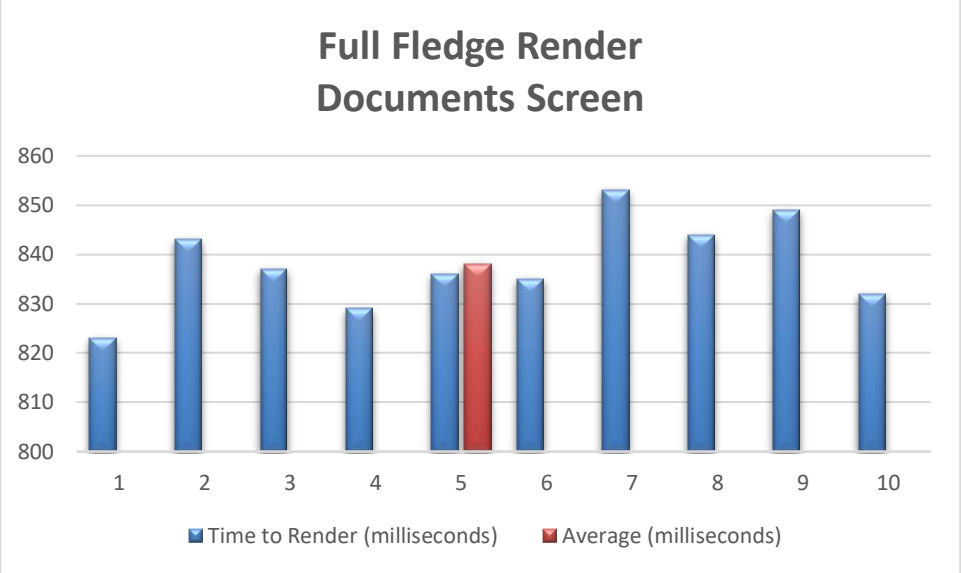

Figure 42 - Full Fledge time to render Documents screen

An ANOVA test was performed on each of these screens and their corresponding metrics, testing the Full Fledge solution data against the original data showcased in Figure 34, 35 and 36. The results of these are shown in Tables 11, 12 and 13:

Table 11 - ANOVA test between Original and Full Fledge solution on Plan screen.

Groups	N	Mean	Standard Deviation	Standard Error
Original	10	3557.7	88.9457	28.1271
Full Fledge	10	733.1	27.3352	8.6441

Table 12 - ANOVA test between Original and Full Fledge solution on documents screen

Groups	N	Mean	Standard Deviation	Standard Error
Original	10	1663.7	41.355	13.0776
Full Fledge	10	415.3	22.9155	7.2465

Table 13 - ANOVA test between Original and Full Fledge solution on illustration screen

Groups	N	Mean	Standard Deviation	Standard Error
Original	10	2831.2	33.4026	10.5628
Full Fledge	10	551.7	17.9137	5.6648

Figure 43 portrays the comparison between each solution in the time taken to render each considered screen in milliseconds:

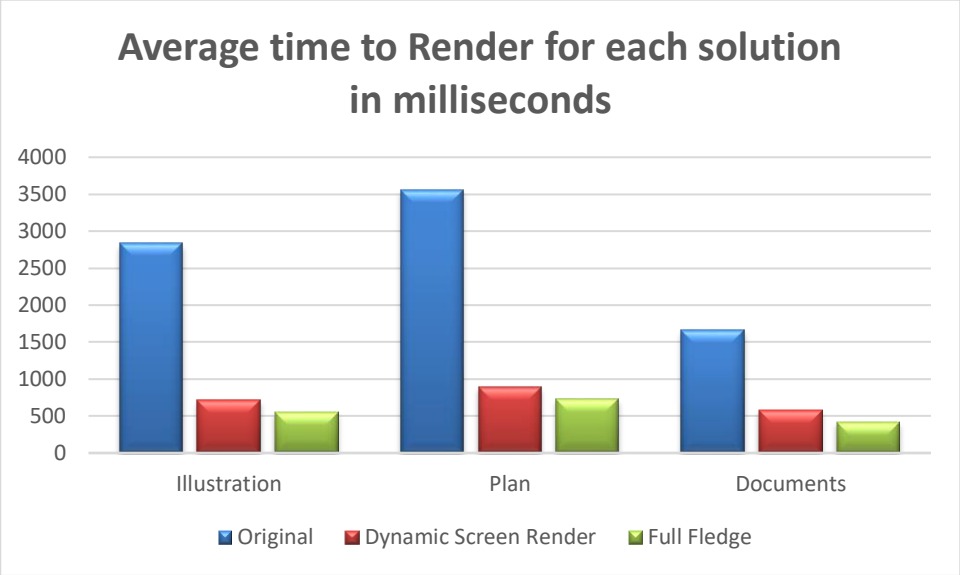

Figure 43 - Average time to render for each solution in milliseconds

Analyzing the data above it's possible to conclude that on average, the improvements percentage wise for each of the screens are as follow:

- Illustration screen – 80,5% improvement
- Plan screen – 79,4% improvement
- Documents screen – 75,04% improvement

According to the Standard Deviation and Standard Error values, with the implementation of the Full Fledge solution the time taken to render is decreased in comparison to just the Dynamic Screen Render. This means that the caching mechanism despite not improving greatly percentagewise compared to just the Dynamic Screen Render, it still does add in average an improvement of 6,593% performance wise.

7. Conclusion

Msg life Iberia is in need of optimizing the performance of the applications that are developed internally, especially in terms of frontend rendering. The current solution for webpage rendering, Render Queue, has some performance issues which result in a particular screen having the necessity to be rendered multiple times with minor changes.

Render Queue does require the interpretation of metadata both at client and server side constantly in order to render a particular screen. This means that interactions with a screen results on modifications within an insurance product metadata, and in order to properly render a screen with the updated metadata, it has to be constantly reinterpreted. This thesis describes three different solutions which work individually, but whose strengths shine when coupled together being them: HTML Template Rendering on an application build pipeline, Dynamic Screen Render Engine and an HTML caching system.

Each of these solutions solves individual problems by themselves, and as a whole, provide a full fledge solution for msg life Iberia capable of replacing the Render Queue functionalities while improving the frontend rendering performance. The HTML Template Rendering component is capable of generating HTML template files which are no less than basic representations of insurance products metamodels. These template files are bundled within the msg.Sales application in order to be used at runtime.

The Dynamic Screen Render Engine is capable of supporting all the internal frameworks used by msg.Sales namely Flow, maintaining the configurability of screens by using the Flow framework, but reducing the overall time taken to interpret metadata and page rendering as with it, the frontend does simply apply the HTML template that is created by the msg.Sales API in accordance to the insurance product definition provided by the PDS and the Flow configurations for all screens.

Last but not least, an HTML caching mechanism which reduces the time spent on frontend/backend communication when a previously rendered screen within msg.Sales is intended to be re-rendered since the frontend directly reuses the cached HTML file and no longer needs to invoke msg.Sales API.

Overall, by coupling and integrating the mentioned components, the performance tests on three different screens within msg.Sales result in considerable improvements which range between 75.04% and 80.5% of time reduction of each rendered screen.

The future steps identified as study progression in regards to theme that this thesis follows is no less than improving the solution in order to handle a PDS whose model is

extremely volatile by rendering the screens dynamically and update its' templates according to the metadata volatility offered by the PDS.

7.1. Limitations & Future Work

The Dynamic Screen Renderer despite being able to achieve the intended behavior, does not contemplate the use of a fully dynamic PDS whose product definition models do change extremely regularly. As such, the solution has to be enhanced in the future in order to support the PDS's which have more volatile product definitions.

Another aspect which is not supported to the extensibility of the framework to partially support runtime template rendering in accordance to metadata volatility. As such, in the future the Dynamic Screen Render needs to be capable of not only creating HTML Templates for screens in the build process, but also be able to generate templates in accordance to small metadata structures, enabling the Frontend to render all the metadata asynchronously and make use of the Cache mechanism for these small generated fragments.

8. References

[1] J.Melbin, "Hierarchical caching techniques for efficient dynamic page generation". Patent 6,397217, 28 2002.

[2] W. S. D. Greene, "Profitability and Efficiency in the U.S. Life Insurance Industry," *Journal of Productivity Analysis,* no. 21, pp. 229-247, 2004.

[3] F. D. Lewis, "Dependents and the Demand for Life Insurance," *The American Economic Review,* vol. 79, no. 3, pp. 452-467, 1989.

[4] W. C. a. Y. T. Cuifeng Huo, Online insurance's business process and marketing decision, Dalian: 2nd International Conference on Industrial and Information Systems, 2010.

[5] m. l. Iberia, *Internal Documentation,* Accessed In, 2020.

[6] R. C. Gronback, Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit, 2009.

[7] V. E, Generative and Transformational Techniques in Software Engineering II, Berlin: Springer, 2008.

[8] P. K. Arie Van Deursen, "Domain-Specific Language Design," *Journal of Computing and Information Technology,* 2001.

[9] P. F. Knights, "Rethinking Pareto analysis: maintenance applications of logarithmic scatterplots," *Journal of Quality in Maintenance Engineering,* vol. 11, no. 4, pp. 252-263, 2001.

[10] S. T.L, What is the Analytic Hierarchy Process?, Berlin: Springer, Heidelberg, 1988.

[11] Y. P. A. S. A. Osterwalder, Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers, Self-Published, 2020.

[12] K. Dewulf, "Sustainable Product Innovation: The Importance of the Front- End Stage in the Innovation Process," *Advances in Industrial Design Engineering,* pp. 139-166, 2013.

[13] A. G. B. S. C. A. F. E. F. S. Koen P, "Fuzzy-FrontEnd: Effective Methods, Tools and Techniques," in *The PDMA ToolBook 1 for New Product Development*, Wiley, 2002, p. 480.

[14] J. A. N. James C. Anderson, "Business Marketing: Understand What Customers Value," *Harvard Business Review,* pp. 58-65, 1998.

[15] R. &. A. H. A. B. &. S. Z. Asgarpour, " A Review on Customer Perceived Value and Its Main Components.," *Global Journal of Business and Social Science Review.,* no. 1, pp. 632-640, 2015.

[16] V. Allee, "Value Network Analysis and," *Journal of Intellectual Capital,* vol. IX, no. 1, pp. 5-24, 2008.

[17] M. Porter, The Competitive Advantage: Creating and Sustaining Superior Performance, New York: Free Press, 1985.

[18] M. W. G. Q. Tu, "The build-time software architecture view," in *IEEE International Conference on Software Maintenance*, Florence, 2001.

[19] R. K. N. Seth, "ACI (automated Continuous Integration) using Jenkins: Key for successful embedded Software development," in *2nd International Conference on Recent Advances in Engineering & Computational Sciences (RAECS)*, Chandigarh, 2015.

[20] B. R. Robert France, "Model-driven Development of Complex Software: A Research Roadmap," IEEE, Minneapolis, 2007.

[21] G. M. Richters M, "Formalizing the UML Object Constraint Language," University of Berlin, Berlin, 1998.

[22] S. H. K. Czarnecki, "Feature-based survey of model transformation approaches," *IBM Systems Journal,* vol. 45, no. 3, pp. 621-645, 2006.

[23] N. K. K. S. G. H. M. M. ,. H. M. Y. Nakano, "Web performance acceleration by caching rendering results," in *17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Busan, 2015.

[24] D. R. C. A. K. a. F. E. B. A. Su, "Drafting Behind Akamai: Inferring Network Conditions Based on CDN Redirections," *IEEE/ACM Transactions on Networking,* vol. 17, no. 6, pp. 1752-1765, 2009.

[25] R. K. S. J. S. Erik Nygren, "The Akamai Network: A Platform for High-Performance," *ACM SIGOPS Operating Systems Review,* pp. 2-19, 2010.

[26] M. F. a. M. Ishizuka, "Semantic Structure Content for Dynamic Web Pages," in *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, Toronto, 2010.

[27] R. K. W. James R. Curran, "Transformation-Based Learning for Automatic Translation from," University of Sydney, Sydney, 2006.

[28] S. H. Krzysztof Czarnecki, "Classification of Model Transformation Approaches," in *Workshop on Generative Techniques in the Context of Model-Driven Architecture*, Canada, 2003.

[29] A. R. F. V. Paolo Ciancarini, "An extensible rendering engine for XML and HTML," *Computer Networks and ISDN Systems,* vol. 30, no. 1-7, pp. 225-237, 1998.

[30] J. R. L. F. L. Neil Young, "Insurance Contract Interpretation: Issues and Trends," *the Insurance Law Journal,* p. 625, 1975.