Software Architecture of an Insurance System based on Services

André Viegas

Instituto Superior Técnico

Abstract. This document presents a Case Study regarding a Transformational programme implemented by an Insurance company, with special focus on its architecture, and the architecture of some of its landscape's systems. This document focus on the challenges presented when the new architectures were designed, in its business and system quality attributes, and how they were instantiated in concrete scenarios. It is also presented the tactics used to achieve the intended quality attributes of the previously defined scenarios. At last, an analysis of the current state of the Insurance company is done, presenting benefits from this Transformational programme, and also new challenges that were raised due to the decisions taken and reported in this document, because in the end, designing software architectures is a continuous, cyclic process, which results in a path that is made by walking.

Keywords: Software Architecture; SOA; ESB; Architectural Quality Attributes

1 Introduction

Insurance companies are pressured to optimize their operations in order to be able to respond quicker to market opportunities and provide the best products to their clients, and do all that in a cost-efficient way.

It was this line of thought that led the Insurance company featured in this Case Study, named InsuranceCorp, into a journey with the goal of optimizing its operation by doing more, cheaper and faster. To do so, the company started a Transformational programme that would led to multiple transformations in its organization.

1.1 Document's purpose

The main purpose of this document is to bring light to the complex process of transforming an architecture in a company with a relevant size, providing a real insight of the challenges and decisions that were taken during the first releases of the Transformational programme, focusing in the Integration work done to achieve those goals.

1.2 Document's structure

This Case Study is divided in 3 phases:

- Context - The first phase of this document provides an overview of the Insurance world, as it presents the circumstantial topics, and also the ones specific to InsuranceCorp;
- Report - The second phase will provide a view of the work done in this Transformational programme regarding the architecture of InsuranceCorp;
- Results - The third and last phase wraps-up this Case Study, providing information regarding the current situation of the Transformational programme and InsuranceCorp, the achieved benefits, the lessons learned and some personal considerations;

1.3 Methodology

The methodology used on this Case study follows the approach suggested by [1], where the architecture cycle is composed by the definition of the Business quality attributes, which will originate the System quality attributes intended for the architecture. Those are the drivers to design the architecture and will be instantiated in scenarios. In order to address those scenarios, the correct tactics are then chosen and implemented. The architectural views presented throughout the Case study base themselves in the content of the book by [5].

1.4 Assumptions and Restrictions

Taken the privacy requirements in this case, some details have been omitted without prejudice to the overall case exposition. These are the assumptions, and also restrictions, due to confidentiality agreements:

- Information that can directly, or indirectly, identify the company will be omitted;
- The real names of the applications/entities that made the company's previous architecture will not be identified;
- Not all the technologies that comprise the new architecture will be identified;
- Details that may compromise the current architecture of the company will not be mentioned;
- No human resources from the parties involved in this Transformational programme will be identified;
- The Integration work will be the only one with a higher level of detail in terms of specification of the architecture, challenges, solutions and lessons learned;
- The architectural representations in this document will not be totally exhaustive in terms of number of systems in InsuranceCorp;

2 Falling Behind

In 2010, InsuranceCorp found itself in a precarious situation: its systems and processes were old and very inefficient. Not only that, but the way it worked, created a lot of entropy and obstacles to all the changes and improvements that the business required. Systems that were developed decades ago, with a multitude of "band-aids" in them through the years, made it very expensive and time-consuming to make the smallest change.

Systems' knowledge was decreasing, due to retiring resources, and the documentation regarding the systems was lacking, which led to a short of in-house knowledge in terms of development and maintenance of those systems. The technology on which they were developed, COBOL on Mainframe, is a somehow scarcer skill nowadays, therefore expensive to hire. This characteristics highly compromised the buildability of the architecture. That increased the time to market and the cost of all maintenance projects, either corrective or evolutionary.

3 Business Qualities - what was intended

InsuranceCorp began by identifying the 3 Business Qualities which provided the biggest drivers for the Transformational programme:

Time to Market:
The easier it is to design and implement new products, the quicker is the time-to-market of it;

Cost and Benefit:
InsuranceCorp were longing for an architecture that would provide the best benefits regarding its costs of implementing and maintaining it;

Roll-out schedule:
Systems in InsuranceCorp were outdated and in their way of being replaced, however its sunset would have to be gradual. The final goal is to have all those mainframe systems completely sunset and replaced with the newer systems;

4 Designing InsuranceCorp's Architecture

Once the Business quality attributes were set, the foundations of the architecture were in place. Now, Business attributes have to be decomposed in System quality attributes, which will determine the expected behaviour and expectations from the systems at a more operational level.

4.1 Easiness to add/modify an Insurance product

Modifiability System quality

InsuranceCorp wanted a system that would allow it to add a new product, or change an existent one, in a quick and easy way.

How it relates with other qualities
This quality attribute that was desired to the System is a logical derivation from the **Cost and Benefit** and **Time to Market** Business Qualities.

Scenario 1 - Insurance Products

Source InsuranceCorp's Business department;
Stimulus Request the addition/change of an insurance product;
Environment Design time;
Artifact Insurance Core System;
Response The roll-out of a new product, or a change, must be possible in a quick manner and with a short budget;
Response Measure The effort it takes in man/hours must be much faster than with the past architecture; The cost it takes to roll-out must be cheaper than with the past architecture;

Scenario 2 - Pricing changes

Source InsuranceCorp's Business department;
Stimulus Reconfiguration of Pricing settings;
Environment Design time;
Artifact Insurance Core System;
Response The roll-out of the new Pricing algorithm configuration must be possible in a quick manner and with a short budget;
Response Measure The effort it takes in man/hours must be much faster than with the past architecture; The cost it takes to roll-out must be cheaper than with the past architecture;

How it was achieved
InsuranceCorp decided to use a market's key Core system off-the-shelf (TNCS - The New Core System), implemented in cheaper and contemporary technology. By using an off-the-shelf system, with its modules, processes and capabilities already pre-defined, InsuranceCorp would greatly increase its Time-to-Market quality attribute that it so long desired.

4.2 New and old systems must be integrated

Interoperability System quality

The architecture had to be designed in a way that would allow all the systems to integrate seamlessly in order to support the business operations, independently of which system is being used - old or new.

How it relates with other qualities
This System quality attribute derives from the **Roll-out Schedule** and **Cost and Benefit** business qualities.

Scenario 1 - Guarantee systems' mutual communication

Source InsuranceCorp's IT;
Stimulus Addition/removal of a system to/from the architecture;
Environment Design time;
Artifact InsuranceCorp's Architecture;
Response All systems continue to work integrated;
Response Measure No system shall lose functionalities due to the removal of another system from the landscape; Cost of adding/removing a system should be minimal;

Tactic applied
InsuranceCorp opted to put in place a Service-Oriented Architecture, with the implementation of an Enterprise Service Bus in order to centralize the communication between components, and orchestrate the interfaces of all systems.

Service-Oriented Architecture
The Service Oriented Architecture (SOA) is an architectural model that stands in the paradigm of Service Orientation, which consists on the creation of self-contained logical units, known as services, which can be used repeatedly, individually or orchestrated together with other logical units, in order to achieve a certain business functionality.

This paradigm requires a much more planned and thought approach upfront when compared to others, since it needs to define the set of operations that will allow other systems to consume. For it to be considered a SOA, all online communications performed between its systems must be done through service consumption. And that can be achieved in 2 ways: Direct Systems Communication, where all systems connect directly between them; Indirect Systems Communication, where a centralized intermediary mediates the communication between all systems.

InsuranceCorp had so many systems in its architecture that required to talk to each other, that it went for a SOA with a central intermediary. That led to the implementation of an Enterprise Service Bus (ESB).

Enterprise Service Bus
The Enterprise Service Bus (ESB) is a middleware component, used in Service-Oriented Architectures, that implements an intermediary element between mutually-interacting systems, which expose a set of loosely-coupled elements (services) called by the systems in need of an action/data. This allows the creation of a single point of contact for all systems, which defines a specific data model for its interactions, with a single technology.

Scenario 2 - Extend the ESB data model for an Insurance product
InsuranceCorp needed to find a Canonical model to be used in its ESB. One of the situations that would test the Canonical model extensibility, and how well it would coup with InsuranceCorp's business, was the creation/modification of

a product in TNCS, as its SOA layer would now provide some more semantic concepts.

Source InsuranceCorp's Business Department;
Stimulus Creation/modification of an insurance product in TNCS - causing new concepts to be exposed in its SOA layer;
Environment Design time;
Artifact ESB;
Response ESB's data model is extended conserving its identity; Effort in extending the data model is limited;
Response Measure Man/hour's effort in the extension work; Number of inconsistencies in the data model;

Tactic applied
In order to find the best standard, InsuranceCorp's IT architects researched the market for the industry best practices, and reached the following conclusion: for the Canonical Data Model it would be used an insurance meta-model created by IBM named Insurance Application Architecture (IAA).

IAA - The Canonical Insurance framework
The Insurance Application Architecture (IAA) is a complete framework for the design, implementation and operation of an Insurance company. Providing a set of inter-related business models required to support the Insurance business([2]), it offers a blueprint for application and data warehouse development, providing business and technical models, based on the set of insurance industry's standards at multiple levels.

In short, what this framework allows, is the creation of the operation of an insurance company from scratch, by abstracting from the core and support systems, and fully aligned with the industry standards and best practices, while reducing risk and saving costs usually spent in IT analysis and design. For companies already running that desire to completely refactor the way they work, IAA offers a full Target Operating Model, which, depending of how far from IAA they are, can become costly to achieve in the beginning, due to the numerous transformations that it may require.

InsuranceCorp decided to focus only in the IAA's Service Models in order to implement the Canonical Data Model for its ESB. To do so, and following IBM indications, the IAA models considered were the Business Object Model and the Interface Design Model.

Applying IAA, a product at a time
The Purpose of IAA in InsuranceCorp was to design and implement services to support the operation, however the implementation of IAA is non-trivial, so it is necessary to guarantee a set of guidelines, strategies, and processes in order to develop a framework that allows repeatability and method in the implementation of IAA in InsuranceCorp. That framework was composed of the following 12 steps:

- Step 1 - Define application that will drive the mapping process
- Step 2 - Gather application's requirements
- Step 3 - Analyse application's requirements
- Step 4 - Identify operations to implement
- Step 5 - Define Service architecture
- Step 6 - Analyse field's business concepts
- Step 7 - Define the Product structure
- Step 8 - Perform functional mappings
- Step 9 - Document technical path in C#
- Step 10 - Document transformational rules
- Step 11 - Implement components and mappings
- Step 12 - Manage changes to mappings

4.3 Modify/Extend the Service Catalogue

Modifiability System quality InsuranceCorp was very focused on developing things faster and cheaper, and with an ESB in place, the modifications in its service catalogue had to respect those same directives.

How it relates with other qualities
This relates with the **Cost and Benefit** and **Time to Market** Business Qualities.

Scenario - Create/Extend a service

Source InsuranceCorp's IT department;
Stimulus New service/extension of an existent service;
Environment Design time;
Artifact ESB;
Response ESB's modification is done with small cost and effort;
Response Measure Man/hour's effort of the work;

Tactics applied
InsuranceCorp created a service's template, with a number of modules, with its responsibilities well defined. That template was named Service Factory.

Service Factory - Service implementation methodology
This template was developed in-house by InsuranceCorp's IT department and was implemented in BizTalk ([4]) and C#. The architecture designed for the implementation of a service, regardless of its complexity, is based on the execution of a single generic orchestration, that would work as the entry point of the requests that originate a service call, while pre-designated modules are loaded dynamically to address the specific behaviour that is expected from each service. The design of Service Factory's template defined 5 different and inter-independent modules: Resolver, Sequencer, Splitter, Aggregator and Mapper.

These modules are interfaces for specialized implementations, each developed for each service. This way, Transformer just needs to understand the type of module it has to load, disregarding of its specialized implementation, adding to the intended modifiability of ESB. It's execution is divided in 3 stages:

- **Sequencing:** Responsible for loading the *Sequencing* module, which has all the logic of the service in terms of sequential steps needed to be done;
- **Debatching:** A single service request may produce multiple paralell calls to some other service. That's why *Aggregator* and *Splitter* modules are loaded in order to transform the original message, and aggregate/enrich the results from the calls;
- **Mapping:** Load the *Mapper* modules responsible of mapping/translating the requests from IAA to the target's data model, and the responses from that target's data model to IAA;

4.4 Hold future load of requests

Scalability System quality

The ESB would need to be built in a way that would be easily scalable as the number of requests that it would deal with increases.

How it relates with other qualities
This System quality attribute derives from the **Cost and Benefit** and **Roll-out Schedule** Business Qualities.

Scenario

Source Migration process;
Stimulus Increase in the amount of requests to be handled;
Environment Runtime;
Artifact ESB;
Response The ESB handles the new load seamlessly;
Response Measure The operational times of the systems do not float regarding the increase of requests they are asked to deal with;

Tactic applied

Microsoft's BizTalk already has a set of capacities that, together with infrastructural configurations, would allow the architecture to withstand the amount of load that it is being asked to handle.

5 Current State

InsuranceCorp already is different, but still not yet as it envisioned itself. Some of the modifications made achieved the intended benefits, but there were some that didn't quite hit the target as it was initially thought, and others that raised issues that were not foreseen.

5.1 Main Improvements Achieved

InsuranceCorp already harvests benefits from its Transformational programme. Half of its Commercial portfolio is already migrated and managed exclusively in TNCS. This alone reduced the processing in the Mainframe, therefore the operational costs are lower.

5.2 Post-Mortem Analysis

It is currently possible to assess what came up short to its goals, and also what issues were raised due to the decisions and implementations performed. Two cases that fall under that situation are the IAA data model, and also the performance registered by some of its services.

IAA's understandability
Modifiability in ESB's data model was seen as paramount, that's why IBM's IAA was chosen, and why the process of implementing it to InsuranceCorp's reality was so meticulous, and it required the use of existing generic concepts in order to keep the model as open as possible in case some other change was needed. However, this ended up creating a very complex, deep, nested-relations filled data model, that had portions so unnecessarily generic and abstract, that eventually grew into a model very difficult to understand.

This is currently an open issue in InsuranceCorp, which is still being addressed. The more backed-up solution is to create a lighter variation of the ESB's data model, that would share the same naming conventions, and be semantically close, but with less nested-relations, and shallower. This "Light IAA" version would be exposed externally, than a set of modules would be implemented in order to map it to the normal IAA data model.

ESB services' performance
With a huge focus on services' modifiability, based on a mainstay of this Transformational programme which was to reduce maintenance costs, one crucial aspect was overlooked: the performance. It was assumed that the performance registered by all services would all be good, and equal throughout the whole service catalogue. However, that was not true, and it caused some issues in some business processes that were based on the execution of some services that could not perform to the expectations. To solve this, the following solutions are being considered:

- Categorize the service catalogue, splitting it in categories based on services' complexity, average response time and size of payload. Then deploy each one of those groups in separate BizTalk clusters, with different timeouts' configurations;
- Identify the services in the service catalogue that handle the biggest payloads, and change its implementation from synchronous to asynchronous;

- Identify business processes, based on services' consumption, that would be better addressed by a batch solution than by an online solution;
- Do not change anything, and try to improve services' quickness by improving the infrastructure;
- Identify special cases, and provide them with an "architectural dispensation", allowing the consumer system to go directly to the back-end systems' services in order to avoid the overhead caused by the ESB;

6 Conclusions

The following are the major lessons learned, and some personal considerations, with the whole process:

- Using insurance products as a main driver for planning a Transformational programme in an Insurance company is a good idea;
- The ESB's canonical model, must take into consideration its understandability, by minding the amount of nested relations, and how deep it is;
- The ESB's data model has to be generic in order to support future extensions in an easier way. However, abstracting so much the model, taking it far way from InsuranceCorp's reality, severely impacted the quality of it;
- The Service Factory is a good framework for creating services in a similar context, provided that the overhead it creates is acceptable;
- Whenever a software has to be designed, the requirements and the quality attributes gathered upfront, are rarely the only ones expected as the system gets in production;
- Some system quality attributes impact other system quality attributes, e.g. modifiability of a system vs. its performance, therefore an architect must mind the trade-off existent in designing a software, and more often then not, it has to arbitrate the prioritization of a system's quality attributes;
- Past experiences and knowledge of best practices are paramount to a good software design, because it will help addressing the expected system's quality attributes, and also help predict future requirements;
- Designing a software's architecture is a continuous, cyclic and ever evolving process;

References

1. Bass, Len and Clements, Paul and Kazman, Rick: Software Architecture in Practice, Addison-Wesley Longman Publishing Co., Inc., Boston (2003)
2. IAA-IIW Introduction and Models, `http://www-903.ibm.com/kr/event/download/200804_337/s337_b02_1.pdf`
3. C# Programming Guide, `https://msdn.microsoft.com/en-us/library/67ef8sbd(v=vs.100).aspx`
4. Microsoft BizTalk Server, `https://msdn.microsoft.com/en-us/library/dd547397(v=bts.10).aspx`
5. Garlan, David and Bachmann, Felix and Ivers, James and Stafford, Judith and Bass, Len and Clements, Paul and Merson, Paulo: Documenting Software Architectures: Views and Beyond, Addison-Wesley Professional (2010)