

Leiden University

ICT in Business and the Public Sector

Formalising Enterprise Architecture Codification through Explicit Reasoning: An LLM-supported Approach to ArchiMate Modelling

Name:	Niels Cluistra
Student ID:	s2628945
Date:	29/07/2026
1st supervisor:	Drs. J.B. Kruiswijk
2nd supervisor:	Dr. A. H. Zohrehvand
Company supervisor:	Drs. O. Stegeman

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Einsteinweg 55
2333 CC Leiden
The Netherlands

Abstract

Background: Enterprise Architecture models provide organisations with insights into their information landscape. However, constructing these models based on unstructured information is an interpretative and time-consuming process of codification. Existing codification approaches mainly support the codification of structured information. Therefore, formalising the codification of unstructured information remains largely unexplored. Large Language Models may offer a solution for the codification of unstructured information, but their performance is uncertain.

Aim: This study investigates whether the reasoning process of Enterprise Architects can be formalised into a codification method. Furthermore, it researches whether this formalised codification method then can be operationalised by Large Language Models to support the construction of ArchiMate models (Enterprise Architecture modelling language).

Method: This study follows the Design Science Research Methodology. We performed interviews with Enterprise Architects to elicit their codification process and derive mapping rules. These findings are formalised into a codification method and then implemented into a Multi-Agent System that makes use of LLMs to execute this method. The designed codification method and tool are evaluated through precision and recall measurements, execution time and expert evaluation.

Results: The findings show that Enterprise Architecture codification is not only a task of translation, but is a structured reasoning process that involves inference, interpretation and representation as well. This reasoning process is formalised into a codification method which could be successfully operationalised by multiple Large Language Models. The resulting multi-agent system is able to generate usable ArchiMate models while it reduces modelling effort as well. Furthermore, the evaluation suggest that the usage of a codification method enables automation and not only the Large Language Models.

Conclusion: This research shows that Enterprise Architecture codification can be formalised into a reasoning process which can be operationalised by AI. The results suggest that the main challenge in Enterprise Architecture modelling lies in the formalisation of expert reasoning instead of automating the generation of the model. These findings provide a theoretical contribution to the literature on Enterprise Architecture codification and provide a practical basis for Enterprise Architecture modelling supported by AI.

Acknowledgements

I would like to sincerely thank my university supervisors Drs. J.B. Kruiswijk and Dr. A.H. Zohrehvand. They supported me continuously, providing feedback and help to make sure that this research could come to a good end. They were always available to help me, which I appreciate a lot.

Secondly, I would like to thank my company supervisor Drs. O. Stegeman. He was always available to discuss my ideas, improve my reasoning and helped me a lot with contacting Enterprise Architects for my interviews. Furthermore, I would like to thank Mr M. Maduro for welcoming me into his team and for providing me with the opportunity to perform my thesis internship at Capgemini.

Thirdly, I would like to thank all the Enterprise Architects who took their time to help me throughout the research. This research only could be performed based on their insights. Even though they all were busy with their assignments, they found time for me and helped me set up this research. Thererfore, I am thankful for their effort.

Contents

Abstract . . . 1
Acknowledgements . . . 1

1 Introduction **3**
1.1 Problem Statement . . . 3
1.2 Research Questions . . . 4
1.3 Research Scope . . . 5
1.4 Overview of the Thesis . . . 6

2 Background and related work **8**
2.1 The Benefits of Enterprise Architecture . . . 8
2.2 The Architecture Development Method . . . 9
2.3 ArchiMate . . . 10
2.4 Large Language Models . . . 11
2.5 Related Work . . . 12
2.5.1 Codification . . . 12
2.5.2 Large Language Models for Codification . . . 15
2.5.3 Limitations of Existing Approaches . . . 15

3 Method **16**
3.1 Identify Problem & Motivate . . . 16
3.2 Define Objectives of a Solution . . . 17
3.3 Design & Development . . . 18
3.4 Demonstration . . . 18
3.5 Evaluation . . . 18
3.6 Communication . . . 19
3.7 Interview and Qualitative Analysis Procedure . . . 19
3.7.1 The Interviews . . . 20
3.7.2 Qualitative Analysis . . . 20

4 Problem Identification **22**
4.1 Interview Results on the Problems with Codification . . . 22

5 Objectives of a Solution **24**
5.1 How Enterprise Architects Codify . . . 24
5.2 Minimal Input Necessary for Constructing an ArchiMate Model . . . 25
5.3 Requirements of the Method and Tool . . . 27
5.3.1 Requirements Elicited in the Interviews . . . 27
5.3.2 Survey Results . . . 27
5.3.3 Final Requirements . . . 29

6 Design & Development **31**
6.1 Codification Method . . . 31
6.2 Evaluation Tool . . . 35
6.2.1 AI Agents and a Multi-Agent System . . . 35
6.2.2 Prompt Engineering of AI Agents . . . 37

7 Demonstration **39**
7.1 Generated Views . . . 39
7.2 How to Work with the Tool . . . 41

8 Evaluation **43**
8.1 Evaluation Results . . . 43
8.2 Evaluation of Problem Resolution . . . 45
8.3 Evaluation of the Design Objectives . . . 45

9 Discussion **47**
9.1 Interpretation of the Findings . . . 47
9.1.1 Codification is more than Translation . . . 47
9.1.2 The Role of Explicit Mapping Rules in EA Codification . . . 48
9.1.3 The Formalisation of Codification enables Automation . . . 49
9.1.4 Validation of the Codification Method . . . 49
9.2 Comparison with Existing Literature . . . 50
9.2.1 Modelling of Enterprise Architecture . . . 50
9.2.2 Codification of Enterprise Architecture . . . 50
9.2.3 LLM-Supported Modelling . . . 51
9.3 Limitations . . . 51
9.3.1 Generalisability of the Codification Method . . . 51
9.3.2 Scope of the Codification Method . . . 52
9.3.3 Design Limitations . . . 52
9.3.4 Evaluation Limitations . . . 53

10 Conclusions **54**
10.1 Answers to the Research Questions . . . 54
10.2 Contributions . . . 55
10.2.1 Academic Contributions . . . 55
10.2.2 Practical Contributions . . . 55
10.3 Future Work . . . 55
10.3.1 Extending the Scope of the Codification Method . . . 55
10.3.2 Validation in Multiple Organisational Contexts . . . 56
10.3.3 Interactive and Iterative Codification . . . 56
10.3.4 Generalising the Codification Approach . . . 56

A ArchiMate Specification **61**

B Topic List Interview: Problem Definition, Objectives of a Solution and Requirements **74**

C Survey Questions **78**

D Codification Method for Evaluation Interviews **86**

E Evaluation Tool for Evaluation Interviews **87**

F ArchiMate view for Evaluation Interviews **89**

G System Prompt Business Layer Agent **91**

H System Prompt Application Layer Agent **102**

I System Prompt Technology Layer Agent **113**

J System Prompt Relationship Agent **126**

K View generated by Claude Haiku 4.5 **134**

L View generated by GPT 5.4 **136**

M Case for the Case Study **138**

Chapter 1

Introduction

In this chapter, we provide the foundations of this thesis. First, the problem statement illustrates which problems with translating processes into ArchiMate models this thesis aims to address. Next, we define the research questions that this research addresses. The research scope dives deeper into the boundaries of this thesis. Lastly, an overview of the upcoming chapters of this thesis is given.

1.1 Problem Statement

The ability to use, create and transfer knowledge may allow the improvement or creation of products. However, knowledge is often tacit, meaning that it often resides in the mind of an individual. Therefore, tacit knowledge is difficult to transfer to other persons by means of speaking or writing [CBB17]. To be able to manage this knowledge, there are two options. These options are personalisation and codification [LN15]. Our research focuses on the latter. Codification is the process of converting knowledge into information that is processable by machines and humans [JMHGH23]. Thus, in order for tacit knowledge to become useable, the knowledge should be codified in a way that is understood, used and stored by the organisation. Codification captures knowledge and makes it more tangible and explicit. However, even though the tacit knowledge is codified, an individual must have knowledge of the model in which the knowledge is codified to be able to decode and use it [OK21]. For the purpose of codification, there are multiple techniques. One of these techniques is the usage of a Conceptual Model (CM) [JMHGH23].

Our research focuses on the codification of CMs. Conceptual modelling enables the reasoning and understanding of tacit knowledge through a visual representation [CBB17]. At the moment, CM is critical to economically highly valuable and knowledge-intensive areas. Developing high-quality CMs is important for the development of high-quality information systems. This is the case as CMs provide a means for the analysis of future systems and for modifying, extending, maintaining or repairing an information system. CMs of poor quality lead to misunderstandings and systems that are designed incorrectly. Therefore, they entail costly re-engineering and recovery efforts later in the lifecycle of the information system [MPR25]. Therefore, one must design a good CM. Examples of CMs are value models, goal models, process models and information system models [DHK+20]. These models are all on individual systems. CMs can describe the whole information landscape surrounding individual systems as well. These CMs are called information landscapes. Within this research, the focus lies on the codification of information landscapes. These models go into processes and the whole information landscape supporting these processes. Enterprise Architecture (EA) is an instance of codification.

As enterprise systems constantly need to be re-engineered to be able to respond to technological evolution and changing market demand, EA is used as a tool to assist stakeholders in managing changes and system engineering [SCC13]. However, some problems arise when using EA. It is common practice for Enterprise Architects to collect information via stakeholder interviews and then manually enter this information into a model [FPB+12]. Therefore, EA is a very time-consuming task [FPB+12] [FBH+13] [TSC14]. Moreover, as EA is a time-consuming task, it is difficult to keep the models up-to-date [Far12]. Thus, collecting the information as well as the modelling of the collected information takes up a lot of time. The modelling of EA can be done in the ArchiMate language. It is important to note that there are multiple

versions of this language. When talking about ArchiMate, this paper is focused on ArchiMate 3.2. This language is published and maintained by The Open Group and is supported by different consulting firms and tool vendors. ArchiMate provides Enterprise Architects with an instrument to analyse, describe and visualise relationships between business domains [Gronda]. It is used to visualise the business (e.g. process steps and actors), application (applications used by the process steps) and technology layer (the technologies supporting the applications) of a full process. However, ArchiMate suffers from overlapping concepts, a lack of refinement between layers and a lack of precision and semantics. Therefore, verification of the models is left to human interpretation; this leads to the fact that the modelling is done inconsistently, as humans interpret the language in a different way [CBO11].

These problems with codification are not only found within EA. Research showed that the internalisation of tacit knowledge is time-consuming [MF12]. Furthermore, a codification problem is that the translation of one language to another leads to translation problems. Translating a model within the Business Process Modelling Notation (BPMN) language to the Unified Modelling Language (UML) is difficult, as not all the elements of BPMN can be directly translated to UML. Therefore, information can be lost during translation [MR09]. This is the same as when you try to translate a Dutch story into English. You can lose some meaning, because not all Dutch words can be correctly translated. Therefore, codification can be hard, as not everything can be correctly codified. Moreover, learning the language in which you need to codify the information takes up a lot of time and is difficult. Thus, not only is the codifying itself time-consuming, but also the learning of the codifying. Because of this, codifying can increase the error rate of the models. As users try to represent or comprehend the components of each language, errors can arise. This could lead to erroneous or incomplete models, which have a negative impact on business [SSA24].

1.2 Research Questions

A new option to counter these problems arose with the surge of Large Language Models (LLMs). Language models predict the surfacing of a given word given the neighbouring words. Language models get their knowledge of words from the statistical analysis of an extensive amount of textual data. These models become large when fed with a vast amount of data, when having a large number of parameters (weights) and when making use of a big amount of computing power to compute the weights of the data [Raa25]. First, research showed that there were positive indications that LLMs can be used in process mining [JSF23] [BSvdA23]. Later on, research has shown that GPT4 (a type of LLM) is able to accomplish Business Process Management (BPM), another instance of CM's, tasks, based on textual descriptions [GAER23]. Berti et al. took it a step further and proved that most models are good at accomplishing BPM tasks (however, the bigger the model, the better) [BKvdA24]. When you 'feed' the LLM with more data, the results are better as well [JSF23]. So even though manual effort remains necessary, LLMs are able to increase the efficiency and accessibility of process modelling [KBSvdA24a]. The knowledge that LLMs can save you time with process modelling and make it more accessible is a good basis for our research, as we propose to investigate how the codification of EA knowledge can be done and whether LLMs can assist professionals by accelerating ArchiMate modelling (business, application and technology layer). This leads us to our Research Question:

RQ *How can Enterprise Architecture knowledge be codified into ArchiMate models, and how can large language models support this codification?*

Research has shown that business-oriented subtypes of CM (e.g. enterprise modelling) are less researched topics within the CM research. This shows that there is a research gap in business-oriented CM that contributes to the shortage of expertise in the workplace [MPR25]. Our study looks into this gap by presenting a method for how the codification of information landscapes can be done and thus how that could be performed more uniformly. Our result increases the understanding of codification, resulting in less experienced individuals being able to do this codification and in LLMs being able to be instructed to perform this codification as well. Furthermore, Sanyoto and Saputra (2024) stated that future research would be research in tools that facilitate the automation or streamlining of the modelling process and that such tools can minimise the challenges there are with codification. In addition to this, they state that the development of an automated translation mechanism is a crucial area for advancement [SSA24]. Our research delves into this gap by looking into the challenges there are with codification and creating a tool that can automatically translate textual descriptions of the information landscape to an ArchiMate model.

Lastly, LLMs can already be used for process analysis, but then specified for value-added analysis [dMCF25]. Furthermore, LLMs are already able to generate process models (but not ArchiMate) based on textual descriptions [KBSvdA24b]. Therefore, this research provides a further understanding of the capabilities of LLMs.

To be able to define an extensive answer to this question, we divide this Research Question into a set of sub-questions. Each answers a sub-problem of this research. The sub-questions all give an answer for a step within the process. This way, we create a theoretical foundation to answer our research question. Furthermore, using this theoretical foundation, we construct a prototype that is able to translate business processes into ArchiMate models.

SQ1 What challenges with the codification of Enterprise Architecture knowledge exist?

This sub-question focuses on the beginning of the Research Question. Using this question, we can establish which challenges there are with the codification of Enterprise Architecture knowledge. Using this information, we have a clear vision of which challenges should then be addressed.

SQ2 What are the mapping rules to codify Enterprise Architecture knowledge?

This sub-question focuses on how the process of creating an ArchiMate model of an AS-IS situation works. To answer this question, we research how these types of mappings are described within existing research. Furthermore, we find out how this can be deduced from the ArchiMate specification and research how Enterprise Architects perform this codification in practice.

SQ3 What requirements do Enterprise Architects have for a tool that translates information landscape knowledge into ArchiMate models?

This sub-question focuses on the requirements for our prototype. We establish these requirements so that we have a clear overview of what Enterprise Architects need in such a tool. This way, we have a solid foundation to design our prototype on.

1.3 Research Scope

This research is done as part of a thesis internship at Capgemini NL, specifically the ADC (Architecture, Delivery Excellence, Change Management) team within the Domains department. The study is carried out over six months and is focused on developing a method on how the codification of information into an information landscape (ArchiMate model) can be done and evaluating this method using LLMs. This method goes deeper into the concepts necessary for this, the way Capgemini architects analyse a process and how these architects translate these processes into an ArchiMate model and the principles of process analysis and Enterprise Architecture. Furthermore, following the proposed method, we provide an end product that accelerates enterprise architecture analysis by translating processes into ArchiMate models.

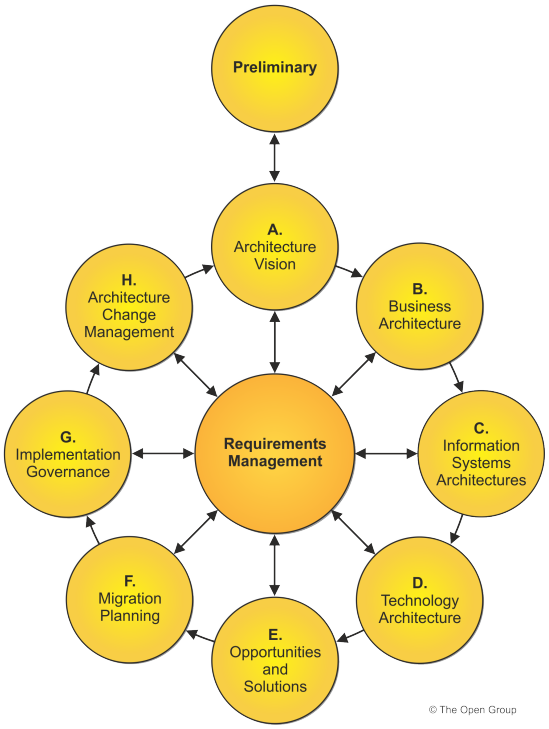


Figure 1.1: This Figure shows the Architecture Development Method (ADM) created by TOGAF

Certain boundaries are defined to make sure the research is feasible. We focus on the Architecture Development Method (ADM) shown in Figure 1.1 created by TOGAF. More specifically, the study focuses on steps B (Business Architecture), C (Information Systems Architecture) and D (Technology Architecture). These steps will be referred to as the three ArchiMate layers during this research. These three layers are business, application and technology. Furthermore, the ArchiMate language is broad and has a lot of elements. This study does not include the business capabilities and the passive elements that ArchiMate has.

1.4 Overview of the Thesis

In Chapter 2, we discuss relevant background information and other work related to this research. It starts by explaining the benefits of performing EA. Then, it dives deeper into ADM by explaining all the steps and specifically defining steps B, C and D. Furthermore, it gives a further explanation of ArchiMate by explaining the elements that ArchiMate has and by explaining the modelling principles of ArchiMate. Moreover, this chapter provides a deeper understanding of the capabilities and types of LLMs. Lastly, we dive deeper into the work related to this research and how our work fills a research gap.

Chapter 3 presents the method we use to conduct this research. This Chapter entails an explanation of the Design Science method developed by Peffers et al. [PTRC07]. Furthermore, it goes into how the Design Science method is used within this research.

Chapter 4 dives deeper into the problem stated in Section 1.1. This chapter presents the findings of the interview with Enterprise Architects on the problems there are with translating business processes to ArchiMate models.

In Chapter 5, we describe what input Enterprise Architects need to construct an ArchiMate model. Furthermore, this chapter presents the findings on how Enterprise Architects make use of LLMs at the moment, what they believe the strengths and weaknesses of LLMs are, and what they need in a tool that helps them with the codification. Lastly, this chapter describes the requirements of the solution. It then makes use of these requirements to find out what the best solution option is for our research.

In Chapter 6, we develop a method for how the codification should be done and then bring this method to life by developing a tool that makes use of LLMs to follow this method. Within this chapter, we make use of the elicited requirements.

In Chapter 7, we demonstrate the method and the tool by demonstrating how the designed method and tool perform in the case study.

Chapter 8 shows the results of the evaluation of the prototype. It shows the results of the precision and recall metrics, as well as the time costs of the method and the evaluation interview on the designs by an Enterprise Architect.

In Chapter 9, we evaluated the findings of this research. Therefore, the results are put into context with the literature. Furthermore, the limitations of this research are discussed.

Lastly, Chapter 10 concludes the research by answering the research questions and summarising the findings. Furthermore, it presents the research's contributions to science and proposes future work.

Chapter 2

Background and related work

In this chapter, we provide the necessary background and an overview of existing academic research that is similar to our research. It builds on the foundation from Chapter 1 by expanding on the concepts of ADM, EA, ArchiMate and LLMs.

We start this chapter by further explaining why organisations make use of Enterprise Architecture. Furthermore, we explain the ADM cycle. We explain the steps within the cycle, focussing ourselves on steps B, C and D, which are within the scope of this research. Then we go into the modelling rules and elements of ArchiMate. In this part, we specify what elements there are within the ArchiMate language and which of those elements are in scope of our research. Lastly, we dive deeper into LLMs, specifying what capabilities LLMs can have and what differences can be seen between types of LLMs.

Lastly, we discuss an overview of related academic work on codification. We first go into what is already done on automatic codification, then we go into similar research that makes use of LLMs for knowledge extraction, and lastly, we further explain what the differences are with our approach and which research gap we go into.

2.1 The Benefits of Enterprise Architecture

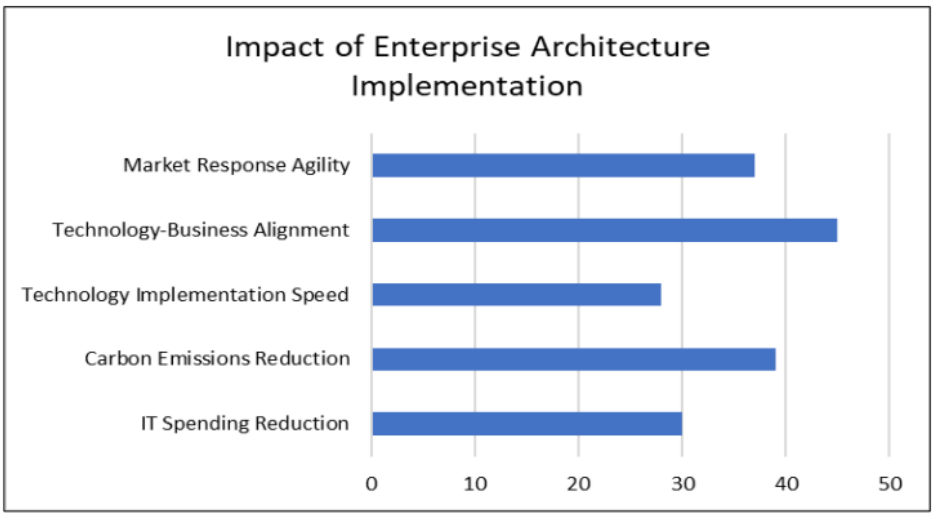


Figure 2.1: This Figure shows the benefits of Enterprise Architecture using Performance Metrics per domain [Kor25]

To successfully integrate new systems into your organisation, companies make use of Enterprise Architecture (EA) to align their IT and Business [Far12] [PCCRP21]. Using both BPM and EA, BPM provides

the business context, metrics and understanding, whereas EA is the discipline for translating strategy and business vision into architectural changes. Therefore, both are necessary for sustainable, continuous improvement [JCO11]. As businesses go through digital transformations, EA is used as a blueprint that guides their strategic decisions, operational improvements and technological investments [Kor25]. Furthermore, EA helps an organisation to standardise its business operations and to incorporate systems throughout different layers (e.g. the ArchiMate layers Business, Application and Technology. This way, companies can achieve organisational benefits and business goals such as business agility, increased value and the creation of competitive advantages [FBH+13]. Figure 2.1 quantifies the impact of EA within an organisation, showing the tangible value [Kor25]. EA is standardised by The Open Group with The Open Group Architecture Framework (TOGAF). TOGAF is the most reliable and prominent standard for Enterprise Architecture. It offers guidance, which enables organisations to operate effectively and efficiently, for example during digital transformations [Grondb]. In conclusion, EA is both a method and a tool that brings a lot of benefits to an organisation, especially when combined with BPM. Using EA, organisations can get more insights into the whole architecture (business processes, application landscape and technologies) they are working in. With these insights, an organisation can make well-considered improvements to their processes, applications and/or technologies.

2.2 The Architecture Development Method

Figure 1.1 shows the ADM cycle. It starts with a preliminary phase; this phase sets the foundation for the architecture work. The main deliverables in this phase are the definition of the architecture principles, the preparation of the organisation for the ADM cycle and setting up the architecture repository. The first phase (Phase A) is focused on developing a high-level vision of the project. It ensures buy-in from your stakeholders and sets the direction for the entire cycle [Das24].

Phase B is primarily focused on developing the business architecture. This architecture defines the business strategy, organisation, key business processes and governance. In this phase, you develop the target architectures and the baseline for the business. Furthermore, you identify the differences between these target architectures and the baseline by performing a gap analysis. Lastly, within this phase, you develop a roadmap which helps the business to transition from the identified baseline to the target architectures. This phase ensures that the business architecture is coordinated with the strategic goals of your organisation [Das24].

Phase C (information systems architecture) consists of two different parts, namely data architecture and application architecture. It is focused on the information systems that are necessary for supporting the business architecture (Phase B). During this phase, you set the baseline and develop the target data architectures, including the data governance, data models and the data management practices. Furthermore, you develop the target application architectures, including the application lifecycle management, the application portfolio and the application interactions. The goal of this phase is to ensure that the information systems are aligned with the needs and objectives of the business [Das24].

Within the fifth phase (Phase D), the focus lies on the technology architecture. It defines the technology infrastructure that is necessary to support the information systems architecture (Phase C) and the business architecture (Phase B). During this phase, an architect develops a baseline as well as target technology architectures. This includes software, network infrastructure and hardware. Furthermore, in this phase, an architect defines guidelines and technology standards which ensure interoperability and consistency. Lastly, a roadmap is developed which helps the business transition from the baseline into the target technology architecture. Therefore, this phase ensures that the technology infrastructure is able to support the whole enterprise architecture [Das24].

The sixth phase (Phase E) determines the solutions and opportunities that are required to reach the target architecture. The goal of this phase is to elucidate a roadmap for implementation and actionable solutions. The following phase (Phase F) focuses on developing a migration plan for the transition from the baseline into the target architecture. This phase guarantees that the transition is planned and managed well. Phase G is focused on the governance of the implementation of the architecture. This way, this phase ensures that the architecture gets implemented as planned and that deviations can be managed effectively. The last phase of the cycle (Phase H) is focused on the management of the changes to the enterprise architecture. This phase makes certain that the architecture remains responsive and

relevant to the altering technology advancements and business needs [Das24].

2.3 ArchiMate

As stated in Section 1.1, ArchiMate is an EA language developed by The Open Group. There are multiple ArchiMate versions; however, this research is focused on ArchiMate 3.2. This language is supported by different consulting firms and tool vendors. ArchiMate provides Enterprise Architects with an instrument to analyse, describe and visualise relationships between business domains [Gronda]. It is used to visualise the business (e.g. process steps and actors), application (applications used by the process steps) and technology layer (the technologies supporting the applications) of a full process. The ArchiMate standard incorporates six primary components:

- Framework: A conceptual framework that allows the classification of architectural phenomena;
- Abstract syntax: This component entails the formal definition of the ArchiMate language. It is a meta-model that provides the characteristics of every language construct as well as its relationships to other constructs in the language.
- Modelling concepts: A set of modelling concepts that allow you to describe the relevant aspects of enterprises (at the enterprise level). This set affirms the abstract syntax and focuses on the concepts and their corresponding meaning, separate from the constructs in which these concepts are used;
- Language semantics: This defines the definition of each relation type and construct;
- Concrete syntax: This syntax explains how the language constructs that are defined within the meta-model are shown graphically;
- Viewpoint mechanism: These mechanisms correlate to the idea of UML diagram types. However, it is more flexible as there is no strict partitioning of the constructs in views [LPJ09]. Using viewpoints, you can design views that correspond to the stakeholders you want to show them to.

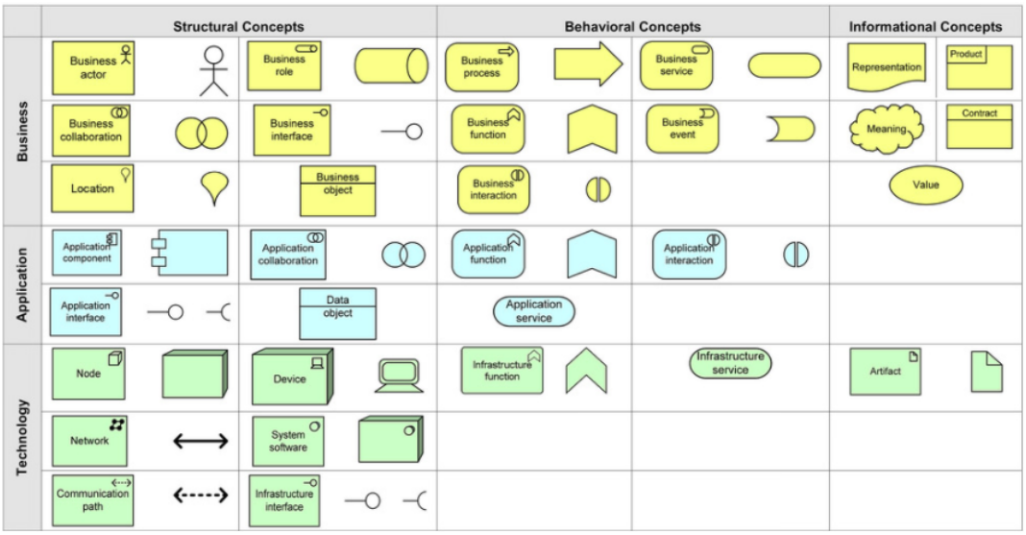


Figure 2.2: The basic ArchiMate elements from the Business, Application and Technology layer

The ArchiMate language consists of relationships and elements. All the relationships and elements are explained in Appendix A. Our research focuses on the Business Layer, Application Layer and Technology Layer Elements as specified in Section 1.3 and makes use of all the possible relationships. The basic elements that belong to these three layers are shown in Figure 2.2.

2.4 Large Language Models

As specified in Section 1.2, Language models predict the surfacing of a given word given the neighbouring words. Language models get their knowledge of words from the statistical analysis of an extensive amount of textual data. These models become large when fed with a vast amount of data, when having a large number of parameters (weights) and when making use of a big amount of computing power to compute the weights of the data [Raa25]. The recent success of LLMs is because of years of development and research of these language models. This research can be seen as four waves, namely statistical language models (SLMs), neural language models (NLMs), pre-trained language models (PLMs) and LLMs [MMN+25].

An SLM sees text as a sequence of words. It estimates the probability of text as the product of the probabilities of its words. NLMs map words to low-dimensional continuous vectors to be able to deal with data sparsity. An NLM predicts the next word based on the aggregation of the embedding vectors of the words that proceed using a neural network. PLMs are task-agnostic, which extends to the fact that a PLM learns the hidden embedding space. A PLM is trained following the pre-training and fine-tuning paradigm. This entails that language models with repeated neural networks or transformers are pre-trained on unlabeled text for general tasks (e.g. word prediction) and then later on fine-tuned to a specific task, making use of a small amount of task-specific data [MMN+25]. These three waves are thus the predecessors of LLMs.

Classification	**Category**	**Description**
Size	Small	Number of parameters $\leq$ 1B
	Medium	1B $<$ Number of parameters $\leq$ 10B
	Large	10B $<$ Number of parameters $\leq$ 100B
	Very Large	100B $<$ Number of parameters
Type	Foundation model	Pretrained language model
	Instruction model	Pretrained and instruction fine-tuned language model
	Chat model	Pretrained, instruction fine-tuned, and chat fine-tuned language model
Origin	Original model	An original model released with either Foundation, Instruction, or Chat model
	Tuned model	Fine-tuned version of an original model
Availability	Publicly available	Model and weights are available due to request to without request
	Publicly unavailable	Model and weights are not publicly available

Table 2.1: Categories of LLMs and their respective definitions [MMN+25]

As shown in Table 2.1, an LLM contains tens to hundreds of billions of parameters. These parameters are pre-trained on a vast amount of text data [MMN+25]. Moreover, Table 2.1 shows the differences between types of LLMs. It shows as well that there can be original models and tuned variants of these models, as well as that the model and weights are not always publicly available. Using these classifications, you can categorise LLMs. For example, the well-known GPT 3.5-turbo model has a large size (20 billion parameters) and is a tuned chat model that is publicly unavailable. Whereas the Claude 2 model has a very large size (137 billion parameters) and is an original chat model that is publicly unavailable [MMN+25].

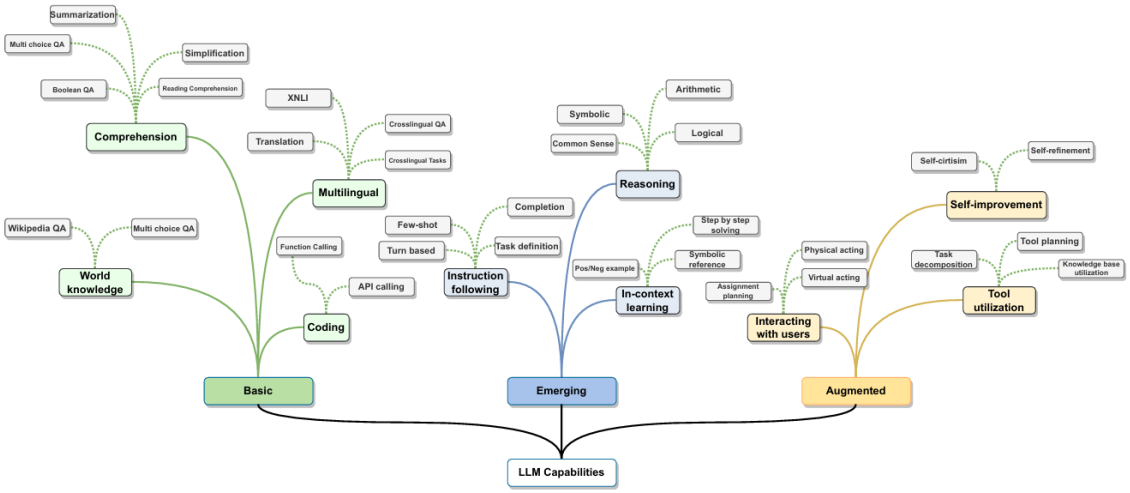


Figure 2.3: The Capabilities of an LLM [MMN+25]

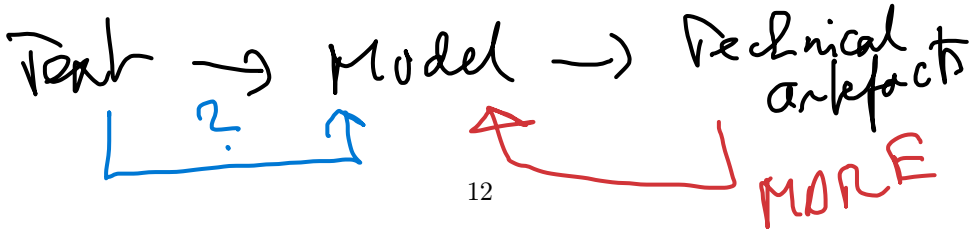
When comparing LLMs to their last predecessor (PLMs), one can see that LLMs are larger in model size and show stronger generation abilities and language understanding. Furthermore, LLMs have the capabilities shown in Figure 2.3. One of these capabilities is that LLMs can learn in-context, meaning that LLMs can learn a new task from a small example set which is presented to them within the prompt (the assignment you give to an LLM). Instruction following is another capability of an LLM; after instruction tuning, LLMs can follow the instructions for new tasks without making use of explicit examples [MMN+25]. The largest advantage of instruction fine-tuning is that no retraining is necessary [ZDL+26]. The last capability we want to highlight is multi-step reasoning. This entails that LLMs are able to solve a difficult task by breaking down that task into reasoning steps [MMN+25]. Using LLMs comes with challenges and opportunities. LLMs can help you understand difficult research; however, LLMs can also carry over the insufficiencies (including errors) that can be found in the training data of the LLM [VBM+23].

The usage of LLMs thus comes with risks. Firstly, applying LLMs to business process analysis raises privacy concerns. In particular, there are concerns about the processing of sensitive organisational data. There is a chance that the inputted process information can be exposed through the retraining of the LLM. Therefore, privacy guarantees and clear data handling protocols are necessary [dMCF25]. Furthermore, LLMs can generate false facts; this is called hallucination [AL26][OSC+25]. Hallucinations can be content that is unsupported by external evidence, or that is factually incorrect. The presence of hallucinations in the output of LLMs undermines the trustworthiness and reliability of LLMs [AL26]. Lastly, LLMs are biased. These biases can come from multiple sources. For example, biases that are encoded in the parameters or architecture of the model, biases that were introduced to the model during deployment or fine-tuning, and biases that were present in the training data of the model. These biases can reinforce inequities, prolong stereotypes and result in unfair outcomes in applications that, for example, make decisions [SNBP25].

2.5 Related Work

2.5.1 Codification

This is not the first research on the codification of models. Most of this research is done on reverse engineering. Model-driven reverse engineering (MDRE) is about discovering models based on legacy artefacts [BCDM14]. The research on ArchiRev is an instance of MDRE. ArchiRev constructs an ArchiMate model based on technical artefacts using deterministic mapping rules. What is further good to notice is that ArchiRev is focused on the application layer instead of the full information landscape [PCCRP21].



	Reverse Eng. Techniques					EA Viewpoints by Layer						
						Business Layer		Application Layer			Technical Layer	
Input IS Artefacts	Static Analysis	Dynamic Analysis	Concept Location	Process Mining	Textual Analysis	Business product/ services	Business process/ function	Application structure/ cooperation	Application behavior/ usage	Information structure	Implementation and deployment	Technical infrastructure
Source code	■	■	■					●	●	●	●	
Execution logs	■	■		■			●	●	●	●	●	●
Database schema	■				■		●			●		
Data model specifications	■		■					●		●		
Enterprise service bus/service interface specifications	■				■	●	●	●	●			
Operational data/ process logs				■			●					
Application configuration file/deployment specifications	■				■			●			●	●

Figure 2.4: ArchiRev's Mapping of Artefacts to Reverse Engineering Techniques and EA Viewpoints [PCCRP21]

Within the ArchiRev method, different software artefacts are seen as input by making use of specific/adapted reverse engineering techniques. These techniques can then discover and model EA elements in ArchiMate. It uses the reverse engineering techniques shown in Figure 2.4 on the artefacts also shown in that Figure. With this combination (technique + artefact), it can then find elements within the Business, Application and/or Technical Layer [PCCRP21].

To codify the artefacts into a model, the method follows three steps. In the first step, the user has to set the list of knowledge sources that are used as input during the process of extracting the elements. Every knowledge source you input is defined by its type, the uploaded artefacts and its associated reverse engineering technique. Thus, one can upload the source code of an application and select that it is a 'Python source code' as artefact type. Then the tool will pick which technique to use. In the second step, the focus lies on the definition of the extraction tasks based on the knowledge sources identified in the first step. Each extraction task can then use one or multiple knowledge sources to generate EA viewpoints. Within this step, you can define the mapping between ArchiMate elements and source code elements. In the third step, the generated views and elements are visualised [PCCRP21].

Algorithm 1. ArchiMate elements generation

```
1  compilationUnitToArchiMateElementMap ← {}
2  annotationToArchiMateElementsMap ← call defaultArchiMateElementMapping ()
3  for each zipEntry in zipFile.entries
4      for each compilationUnit in zipEntry.compilationUnits
5          for each annotation in compilationUnits.annotations
6              type ← annotationToArchiMateElementsMap.get (annotation)
7              if type is not null then
8                  element ← call createNewArchiMateElment (type)
9                  compilationUnitToArchiMateElementMap.put (compilationUnit, element)
10 return compilationUnitToArchiMateElementMap
```

Figure 2.5: Algorithm 1 of ArchiRev that generates ArchiMate elements [PCCRP21]

ArchiRev makes use of a parser which recognises the structure of a part of the source code and then generates an abstract syntax tree of this. From this tree, you can gather information, for example, callable units that belong to a compilation unit. Regarding the codification of ArchiMate elements, this parser detects all the found compilation units and their corresponding callable units (respectively, classes and methods in an object-oriented language such as C# or Java). In combination with these found core elements, the parser then detects the annotations that are made by each class. Annotations are compilation directives that change the behaviour of classes, which provide a lot of semantics for application behaviour/structure views. Together with these annotations, the names of classes are analysed as well

using textual analysis. All the information together is then used to transform as many classes as possible into specific ArchiMate elements [PCCRP21]. This process is described in Figure 2.5.

Compilation Unit Annotation	ArchiMate Element
ManagedBean	Application function
Controller	Application function
Component	Application function
Named	Application function
Service	Application function
Entity	Data object
Table	Data object
MappedSuperclass	Data object
Repository	Application component
SpringBootApplication	Application component

Figure 2.6: The default mapping used in ArchiRev between compilation unit annotations and elements in ArchiMate [PCCRP21]

	Application Function	Application Component	Application Service	Data Object
Application function	Triggering	Triggering	Access	Access
Application component	Serving	Serving	Access	Access
Application service	Realization	Realization	Triggering	Access
Data object	Access	Association	Access	Composition

Figure 2.7: The default mapping used in ArchiRev for relationships between generated elements [PCCRP21]

Figure 2.6 shows the default mapping that is used within ArchiRev that maps the found annotations into Application layer elements. This way of working can be used because all the annotations are found because of the structured data types that are entered. However, as we want to codify unstructured data as well, this way of mapping the elements is not the best way forward, as we do not have a small subset of annotations and elements to map. We want to codify the Business and Technology elements as well. However, the way of mapping used in Figure 2.7 can be used by us as well; by following this way, we can clearly instruct which relationships need to be used between which elements.

```
<RulesDocument> ::= {<Rule>}
<Rule> ::= [<Loop>] {<Condition>} {<Detection>}
<Loop> ::= "for" <HelperGet> ":"
<Condition> ::= "if" <HelperBool> ":"
<HelperGet> ::= "getClasses()" | "getMethods()" | ...
<HelperBool> ::= "isAnnotatedWith()" | "isExtending()" | ...
<Detection> ::= "detectComponent()"  | "detectRole()" | ...
```

Figure 2.8: A simplified form of the conceptual framework for the mapping rules [KKPK23]

Another research on codification is also based on mapping rules. Within this research, they extracted the structure that represents the system based on software development artefacts. This structure then includes how a software system is split up into components and shows what the dependencies among these components are as well. Furthermore, it shows the deployment of these components to nodes. To do so, mapping rules are defined in this project using the language Xtend. A simplified version of these rules is shown in Figure 2.8. Again, this research needs structured input information for the codification. Using these set rules, different models can be codified from the input, such as UML and Palladio Component Models [KKPK23]. Within our research, not all data is structured, but we also perform the setting of rules to base the codification on. These rules are then a basis to work with. Thus, MDRE is possible, but it has to make use of structured data to get results.

2.5.2 Large Language Models for Codification

LLMs are able to understand unstructured data [TS26]. Furthermore, LLMs show strong usefulness and capabilities in generation tasks and broad language understanding [MMN+25] [ZLCB24] [KBSvdA24a] [KBSvdA24b]. It is already possible to codify textual input into process steps using LLMs. Research shows that LLMs are able to understand textual input and turn this input into process steps using a prompt with clear instructions on how to break up an activity into multiple steps. However, this research then goes further into analysing the value added to the activity by each of these steps [dMCF25]. Furthermore, textual input can be codified into a process diagram. This is done in two steps. First, an LLM is used to plan what the diagram should look like. The LLM identifies three components, namely the entries (list of objects), the relationship between the entries and the layout. Using the generated plan, an LLM is called again to generate the diagram [ZLCB24]. Lastly, a tool has already been created which allows its users to describe a workflow in natural language and then codify this natural language into a formal process model using LLMs [KBSvdA24b]. Even though these research studies are mainly focused on process steps and models, they do give evidence that LLMs have the potential to also be able to codify natural language into an information landscape, as process steps are, for example, a part of the information landscape. However, the risks named in Section 2.4, need to be held in account.

2.5.3 Limitations of Existing Approaches

As specified in Section 2.5.1, the approaches on codification make use of structured input. However, within EA, a lot of information is gathered by doing interviews [FPB+12]. The data that comes out of the interviews is unstructured; thus, cannot be codified by the approaches from Section 2.5.1. This is because these approaches are based on deterministic mappings [PCCRP21][KKPK23]. Section 2.5.2 shows that there are studies that show that LLMs can understand textual input and codify this input into process diagrams [dMCF25][KBSvdA24b]. Furthermore, as LLMs can understand unstructured data, it is plausible that LLMs can understand the unstructured data that comes out of interviews and translate this into an information landscape in ArchiMate.

At the moment, there is no research that looks into the codification of full information landscapes based on unstructured data. However, research shows that the development of tools that are able to do so is a crucial area for advancement [SSA24]. Therefore, basing our research on the existing approaches on codification and on recent developments in LLMs, we research how the challenges there are with codification can be addressed by creating a method for performing this codification and evaluating this method using LLMs.

Chapter 3

Method

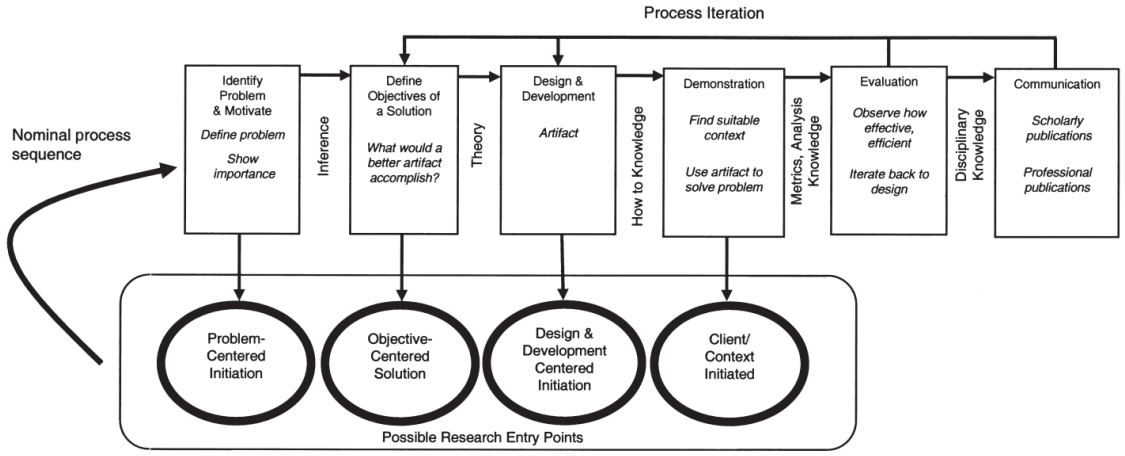


Figure 3.1: This Figure shows the Design Science Research Method process [PTRC07]

The main research method in this study is Design Science. This research method is oriented on the creation of successful artefacts [PTRC07]. Therefore, Design Science fits our research the most, as we are researching the design of a method on how to perform the codification of unstructured information landscape information and then executing this method via an LLM. As shown in Figure 3.1, the Design Science Research Method consists of the following process phases:

1. Identify Problem & Motivate
2. Define Objectives of a Solution
3. Design & Development
4. Demonstration
5. Evaluation
6. Communication [PTRC07]

How our research goes through these process steps and what these process steps entail is explained in the upcoming sections of this Chapter.

3.1 Identify Problem & Motivate

Within this phase, the research problem is defined and it is justified whether there is a value in providing a solution for it [PTRC07].

The problems are identified in Section 1.1 via a literature review. This literature review goes into what codification entails, how Enterprise Architecture is a manner of codifying an information landscape into an architecture language and which problems there are with codification. This then led to the Research Question proposed in Section 1.1. We justify researching this problem by explaining its value in Section 2.5.

To verify whether the problems identified in Section 1.1 exist in the real world as well, we perform interviews with professional Enterprise Architects. Within these interviews, we verify the identified problems and aim to find out whether there are more difficulties with the codification of information landscapes. We make use of interviews as the interview technique helps participants by giving them the flexibility to explain the problems. Furthermore, during the process of an interview, we can collect data that is as detailed as possible. This helps us to provide diversity within the data and to deepen the data [Dur23]. Using a qualitative analysis, we present the findings in Chapter 4. These findings, combined with the literature review, then answer Subquestion 1.

3.2 Define Objectives of a Solution

Within this phase, the objectives that the solution needs to adhere to are defined. These objectives can be inferred from the problem definition and from knowledge of what is feasible and possible [PTRC07].

To define the objectives of a solution, we take three steps. First, we perform a literature review to find out what information is necessary for Enterprise Architects to construct an AS-IS ArchiMate model. Using the interviews mentioned in Section 3.1, we aim to verify these findings and to find out whether there is more information necessary for the codification. Moreover, using these interviews, we aim to discover how Enterprise Architects codify this information into an ArchiMate model. We use the interview technique for these questions for the same reasons as described in Section 3.1.

Furthermore, we elicit the requirements of our design during the interviews. The interview is an unstructured interview, which is the most flexible method that can be used in the process of requirements elicitation. It requires the lowest amount of time to report the requirements and requires little time from the respondents to prepare themselves. This is useful in our situation as the respondents all had assignments they were working on and therefore did not have a lot of time to prepare themselves for the interviews. Another advantage of the usage of unstructured interviews is that even though the number of identified requirements is lower than with other methods, the level of completeness is not that low. This entails that other methods can identify more requirements, but many of these requirements have no part in the solution and therefore are no requirements [RPD20].

The found requirements fall into two different categories. Firstly, requirements on how the tool should perform the codification. Thus, what are the functional specifications for the process of translation. Secondly, requirements on the usefulness of the tool. This way, we aim to discover in what way we can automate the codification of information and what features and information are required for the Enterprise Architects to be able to make good use of our end product. Lastly, using these requirements, we perform a literature review to find out which solution options there are and, most importantly, which solution option best fits our research.

Lastly, a survey is sent to the Enterprise Architects working at Capgemini. Within this survey, they are asked about their experiences with LLMs. More specifically, they get questions on their experiences on how they have used LLMs to translate textual descriptions of the information landscape into ArchiMate models, as well as questions on why they think that what they did worked or did not work. Using these experiences, extra requirements can be set on what architects want to see and how they want to be able to use a tool that can develop ArchiMate models for them. The result of this phase is that we have a clear overview of the objectives of the solution. These objectives consist of solving the problems stated in Section 1.1, of what input is necessary and of which requirements there are for our design.

3.3 Design & Development

Within this phase, the artefact is created. A design research artefact is any designed object that has a contribution to research embedded in its design [PTRC07].

Within this design science, we construct two different artefacts. The first artefact we develop is the main artefact, which is a method on how to codify. Thus, meaning a method on how one could map retrieved information into an ArchiMate model. To develop such a method, we use the knowledge we retrieved in Chapter 5 on what the input for an ArchiMate model should be and on which steps Enterprise Architects take when codifying. Moreover, we make use of the definitions of ArchiMate elements in Appendix A to get a clearer definition of what information should be mapped to which kind of element. Lastly, we perform an interview with an Enterprise Architect to get further details on what signals within documentation or interview transcripts he sees to determine which elements need to be modelled based on the information.

To evaluate this method, we develop a second artefact. This is a tool that brings the first artefact into action using an LLM. Thus, the prototype follows the codification steps that we constructed. This way, we can show whether it is possible to automate codification based on the codification method we propose and thus solve the challenges there are with codification, with the caveat that even though the proposed method can be right, an LLM may not be able to follow this method yet.

Using the requirements elicited in the interviews and survey that are stated in Chapter 5, we construct the artefacts. The construction is done in sprints. To ensure that the model adheres to the requirements and addresses the challenges there are with codification, the created method and tool are evaluated by Enterprise Architects after every sprint. Their direct feedback can then be used to further improve the artefacts. This process ensures that the design is thoroughly evaluated. Furthermore, after every sprint, new requirements on how the codification method should work and on the usefulness of the tool can come up. The result of this phase is that we have a design which follows the requirements we have elicited and that can be used to solve the challenges there are with codification. After this phase, we can give a clear answer to Sub-questions 2 and 3.

3.4 Demonstration

Within this phase, the use of the artefact is demonstrated. It shows whether it can solve the problem or an instance of the problem. This could involve the usage of the artefact in a case study [PTRC07].

In the Demonstration phase, we make use of an anonymised case study provided to us by Capgemini. This case study is an anonymised training case of Capgemini, where the trainees needed to make an ArchiMate model of an information landscape. This content is therefore very useful to us, as using this content, we can demonstrate how well our codification method works. The result of the codification and how we made use of the tool is shown in Chapter 7. As the tool makes use of one LLM to perform the codification method, we demonstrate the generalisability of the codification method by showing that using another LLM leads to the same results. The result of this phase is that we have a clear demonstration of how our method works and how LLMs perform this method.

3.5 Evaluation

Within this phase, it is observed and measured how well the artefact is able to support a solution to the defined problem. Furthermore, the objectives of a solution defined in Chapter 5 are compared to the observed results in Chapter 7 [PTRC07].

The evaluation of the two artefacts is done in two ways. Firstly, we score the generated ArchiMate model on the metrics precision and recall, as well as time and monetary costs. To perform this evaluation, we received a case study from Capgemini as well as the answer model of this case study. We improved this answer model so that it is more real-world-like and then validated this answer model during the interview with an Enterprise Architect, in which we set up the codification method and the answer model. The generated models are evaluated based on the answer model. All elements that are

generated are compared to the ones in the model, and if they have the same element type and the names are (semantically) the same, the generated element is counted as a true positive. So, for example, in the answer model there is an application service element named 'Automated Underwriting Service' and in the generated model there is an application service element named 'Underwriting Service'; then these element types are the same and the names are semantically the same. Therefore, this element is counted as a true positive. The same goes for technology services named 'Deployment' and 'Hosting'. However, when names are semantically the same and element types are not, or element types are the same, but names are not, then the generated element is counted as a false positive. This way, all elements are evaluated. When the elements are evaluated, we look at the relationships between the elements that are counted as true positives and check whether these relationships are within the answer model between the semantically same elements as well. Only the same relationships are counted as true positives. So if an actor is connected via a business service to a business function, but should be directly connected to the business function, the relationship is counted as a false positive. This way, we went over all the relationships between the elements that are counted as true positives.

Furthermore, we perform interviews with Enterprise Architects. These architects were shown our codification method and then a demonstration of the tool. So that they can see how the method works with the tool and the outputted ArchiMate model. After demonstrating this, the Enterprise Architects are asked questions that evaluate the output based on how well they believe the method works, on whether the tool applied these rules correctly, on how well the model reflects the described situation in the input, and on the usefulness of the tool. This is done after the first sprint to gain insights on the progress and get clarity on what else needs to be done to further improve the pipeline. After the second sprint, this evaluation interview is performed as well, but in that case the improvements are for future work or for later improvement of the tool by Capgemini. As we propose to do two sprints, we go through this loop of Designing, Demonstrating and Evaluating two times to make sure our end product solves the challenges there are with codification as well as possible. The result of this phase is that we have a clear evaluation showing the improvement points of the design. These improvement points can be used to further develop the prototype during sprint two. After sprint two, the potential improvement points that still exist can be used as input for future research on this problem.

3.6 Communication

Within this phase, the problem and its importance, the utility and novelty of the artefact, its design and effectiveness are communicated to researchers and other audiences, such as practising professionals, when appropriate [PTRC07].

The research will be communicated by writing a master's thesis about it and giving a thesis defence at Leiden University. Furthermore, we perform an end presentation on this research at Capgemini as well. This way we can also communicate the design to practising professionals.

3.7 Interview and Qualitative Analysis Procedure

As stated in Sections 3.1 and 3.2, the research relies on qualitative interviews to validate the problem and minimal necessary input and to elicit the codification method and requirements for both the codification method and evaluation tool. Furthermore, as stated in Sections 3.3 and 3.5, we performed interviews for setting up the codification method and evaluating it. These evaluation interviews were conducted with five Enterprise Architects, one for setting up the method, three for the mid-term evaluation and one for the final evaluation. These evaluation interviews were not qualitatively analysed. Instead, we constructed the codification method based on the first by going over every element and writing down all the input of the Enterprise Architect within the method and wrote down all the comments on the codification method, evaluation tool and generated ArchiMate model from the other four Enterprise Architects. Therefore, this section describes how participants were selected and how the interviews were conducted for both types of interviews. Furthermore, the section describes how the first type of interviews was transcribed and how the qualitative data analysis for these interviews is performed.

3.7.1 The Interviews

Interview purpose	Number of participants	Outcome
Problem identification	15	Problems with the codification of EA knowledge
Eliciting the codification process	15	The process an Enterprise Architect goes through from starting a project to constructing the ArchiMate model
Eliciting the minimal input necessary for codification	15	The minimal knowledge an Enterprise Architect needs to have to construct an ArchiMate model
Eliciting requirements	15	The requirements on both the codification method and the evaluation tool
Setting up the codification method and validating the answer model	1	Hints and signals to finding the elements we aim to codify a validated ArchiMate answer model
Evaluation of the designs	3	Feedback on the codification method, evaluation tool and generated model to improve the designs in sprint 2
Final evaluation of the designs	1	Feedback on the codification method, evaluation tool and generated model

Table 3.1: The interviews with purpose and number of participants

In total, we interviewed 18 different Enterprise Architects. For the qualitative interviews, 15 Enterprise Architects participated. All participants work as Enterprise Architects at Capgemini. All but one participant have multiple years of experience with this line of work and the ArchiMate language. The last participant works as an Enterprise Architect, but does not have all the certificates for working with the ArchiMate language. We selected these participants via supervisor recommendations and referrals. This means that all Enterprise Architects that we interviewed were either recommended to us via our company supervisor or recommended by the Enterprise Architects that we interviewed. For the interview in which we set up the codification method, we selected a participant from the initial qualitative interviews. This decision was made as this participant was already familiar with the context of the research. This allowed us to build upon the earlier findings and explore the reasoning of this Enterprise Architect in further depth, which resulted in explicit mapping rules for our codification method.

During the evaluation of the designs, three Enterprise Architects, who did not participate during the initial interviews, were interviewed. We chose to interview these participants because this way we reduced the risk of simply confirming the ideas of the original participant group. Instead, the designs were evaluated by Enterprise Architects who had not influenced the development of these designs. This provided a fresh validation of the clarity, applicability and completeness of the designs. During the final evaluation, an Enterprise Architect, who had already participated during the first interviews, was interviewed. We chose to do so as this Enterprise Architect had previously described their practices and therefore could evaluate whether the designs captured the reasoning that was previously described.

As specified in Section 3.2, we made use of semi-structured interviews for the first interview round. The Topic List for these interviews can be found in Appendix B. The interviews took 30-60 minutes and were performed online via Microsoft Teams as well as in-person at the office of Capgemini. During 13 of the 15 interviews, we were allowed to record the audio and make transcriptions of the interviews. During 2 of the 15 interviews, we were thus not allowed to do so and had to make notes during the interviews. Even though the Topic List already shows possible follow-up questions, we were not limited to these questions and were able to ask further follow-up questions as well. All these interviews were transcribed and anonymised before analysis and served as the source for qualitative coding. The last five interviews were simple expert feedback interviews that helped us set up the codification method and evaluate the design. These interviews are not transcribed, as all comments are directly visible within the initial codification method (Appendix D) and evaluation results (Tables 6.1 and 8.3).

3.7.2 Qualitative Analysis

To perform the qualitative analysis, we took the following steps. First, we read the interview transcripts. Secondly, during the reading, we identified which fragments best answered the questions we had and assigned codes to these fragments. Thirdly, we compared the codes across all the interviews and merged similar codes into one. These codes were refined throughout the coding process until they consistently represented the data of the interviews. This led to a final codebook. Furthermore, based on the codes, we created code groups where these codes would then fall in. This way we could categorise all the given codes so that it became clearer which codes represented answers to which questions. A code could only be used once per interview. So when a respondent states that 'there are a lot of different interpretations'

and later on mentions that 'everyone models differently', the code 'ArchiMate is open to interpretation' is used only once to make sure the data did not get skewed. Therefore, the occurrence rate we use in Tables 4.1, 5.1, 5.2, 5.3 and 5.4 means that the code occurred in a certain number of interviews. This certain number is then the occurrence rate. The occurrence rate is used as we have only interviewed 15 participants. Therefore, it is not possible to perform a good statistical analysis on these results.

All identified codes are reported in these tables so that a complete overview of the results is provided. However, for Tables 4.1, 5.1 and 5.2, only codes with an occurrence rate of at least five were taken into account, which is one-third of the sample. This threshold is chosen to focus the analysis on the claims that are supported by a substantial portion of the Enterprise Architects. Furthermore, this threshold reduces the influence of less frequently mentioned observations. For Table 5.3, this threshold would lead to few results. Therefore, we chose to take all requirements on the codification method into account, as it is the main design. Furthermore, as most interviewees only thought of the requirements of the tool and not of the requirements of the method, we believe that the requirements on the method are shared ideas within the Enterprise Architects of Capgemini, but they mostly only thought of the requirements of the tool. Therefore, the requirements with an occurrence rate of 1 out of 15 are still used as final requirements. Moreover, for the requirements of the tool, we set a threshold of three or higher to make sure there were a bit more requirements to work on, while making sure as well that these requirements are backed by multiple Enterprise Architects. Lastly, for Table 5.4, we chose to take all the viewpoints into account with an occurrence rate of two or higher, as otherwise we would have few results and this way we make viewpoints for the B, C and D steps of the ADM, which fits our scope.

Chapter 4

Problem Identification

Within this chapter, we look back at the difficulties there are with codification found in the literature. As explained in Section 3.1, we aim to verify these findings using interviews. The results of these interviews are presented in this chapter using a qualitative analysis.

4.1 Interview Results on the Problems with Codification

To find out which problems exist with codification, we performed a literature review and did interviews to find out which problems exist when codifying Enterprise Architecture knowledge into an ArchiMate model. As specified in Section 1.1, the following difficulties exist with the codification of information landscapes:

1. It is difficult to translate one language into another language as not all elements can be directly translated [MR09];
2. The ArchiMate language is open to the interpretation of the user, leading to different usage of the language between users [CBO11];
3. It is hard to keep your codified model up-to-date [Far12];
4. Codification is a very time-consuming task [FPB+12][FBH+13][TSC14][MF12].

During the interviews with Enterprise Architects from Capgemini, we have tested these difficulties to find out whether these problems are experienced during the work as well. Furthermore, the Enterprise Architects were asked whether they experienced other difficulties with codification.

Problem	**Occurrence in interviews**
ArchiMate is open to interpretation	12
Difficult to codify knowledge to model	7
Codifying takes up much time	6
It is difficult to keep codified models up-to-date	6
People in the organisation see things differently	4
The knowledge is not in the organisation (anymore)	2
ArchiMate repositories contain double elements	2
You do not know everything that you need to know	2
ArchiMate is a difficult language	1
Difficult to keep the model understandable	1
One model should fit multiple perspectives	1
Organisations have their own codification dialect	1
Hard to get people to make time for you	1

Table 4.1: This Table shows all difficulties mentioned within the interview, with the occurrence rate stating how many interviews this difficulty is mentioned in

How we performed and coded this interview can be found in Section 3.7. As explained in Section 3.7.2, the minimal occurrence rate for Table 4.1 is five. Based on this, the problems we take into account from the interviews are:

1. ArchiMate is open to interpretation;
2. It is difficult to codify knowledge to a model;
3. Codifying takes up much time;
4. It is difficult to keep codified models up-to-date.

These problems are the same as the problems found in the literature. When looking at the Topic List in Appendix B, this could be because (with three of the four problems) we asked whether the Enterprise Architects recognise these problems. However, these problems were named as well when we just asked what they believed were the problems with codification. As these problems are the same as the problems found in the literature, we conclude that these four problems are the main challenges within the codification of Enterprise Architecture knowledge to ArchiMate models.

Chapter 5

Objectives of a Solution

Within this chapter, we make use of the interviews for which the topic list can be found in Appendix B to specify the steps that Enterprise Architects take to go from knowledge to an ArchiMate model. Moreover, we perform a literature review to find out what information is necessary for Enterprise Architects to construct an AS-IS ArchiMate model. We then verify and expand these findings using the interviews we had. Furthermore, we present the elicited requirements for our design and method. Lastly, we analyse the survey we have done amongst Enterprise Architects to find extra requirements for the design.

5.1 How Enterprise Architects Codify

Codification step	**Occurrence in interviews**
Talk to/interview people	10
Make codification models to use in conversations	8
Set the scope of the information landscape	8
Iterate over models to gradually improve them	6
Documentation is mostly outdated	5
Go through documentation	4
Perform workshops for information retrieval	3
Check the maturity of the existing architecture	2
Find reference architectures	2
Find out who the stakeholders are	2
Gather information	2
Find bottlenecks within the process	2
Start with constructing a metamodel	2
Translate your findings to ArchiMate	2
Write down the knowledge you elicited	2
Check the bigger picture in which your information system lies	1
Construct process models (e.g. BPMN)	1
Describe the problem you are working on	1
Determine how to use ArchiMate	1
Find out what applications are running	1
Insert the legally obliged process steps in the architecture	1
Get help for the gathering of information	1
Make assumptions based on your experience	1
Match the model you show to the stakeholder you talk to	1
Think of the components and the logic between them	1
Turn the existing drawings into ArchiMate models	1
Follow a top-down approach	1

Table 5.1: This Table shows all codification steps mentioned within the interview, with the occurrence of how many interviews this step is mentioned in

How the coding in Table 5.1 is performed is explained in Section 3.7. Table 5.1 shows all the different steps mentioned in the interviews that Enterprise Architects take to gain knowledge of the information landscape and get to an ArchiMate model. Using these findings, we find a way of working.

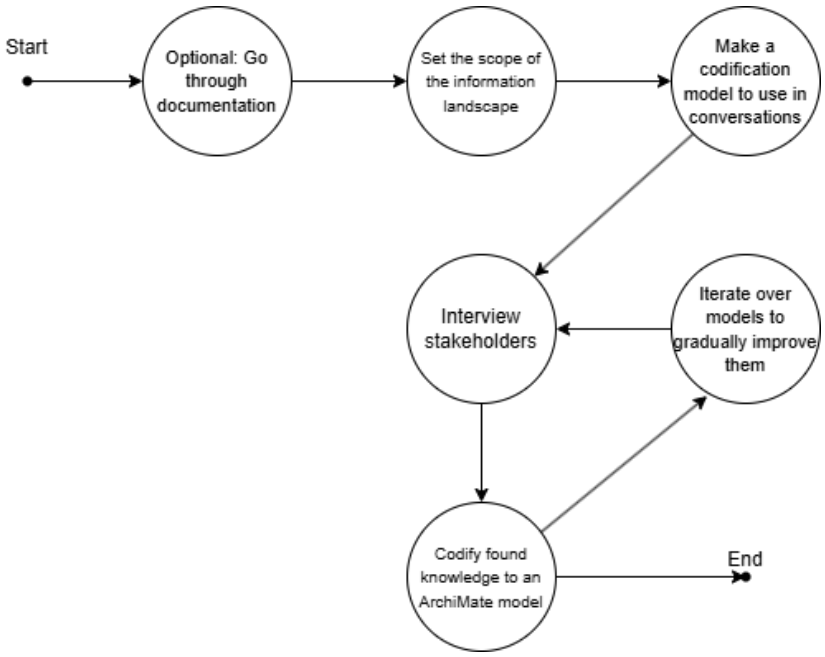


Figure 5.1: The modelling process based on the codification steps from Table 5.1

Figure 5.1 shows a way of working of Enterprise Architects at Capgemini. As stated in Section 3.7.2, we only took process steps of Table 5.1 with an occurrence rate of five or higher into account (the one with a score of four, we made optional) to make sure that at least one third of the Enterprise Architects actually perform this process step. As the steps in Table 5.1 are not in order of when the step is performed within the process, we have put the steps in a logical order. It starts with the optional step of going through the documentation. This step is optional, as only four Enterprise Architects deemed this as full and five others stated that it may not be very useful, as documentation can be outdated a lot. Therefore, the first step is to set the scope of the information landscape. Within this step, you decide what is in scope and what is not. This helps you with setting expectations, working efficiently and with a clear scope at the beginning, you can check whether you fulfilled the assignment at the end. After setting a scope, Enterprise Architects create a codification model (e.g., a simple drawing or a BPMN diagram) that they can use in their conversations. This first model then looks like how they think the information landscape looks based on experience or the documentation. This helps them in the interviews with the stakeholders, as using this model, they have a talking point on which the stakeholders can shoot. During the interview phase, they gain knowledge of the information landscape, which can then be codified into an ArchiMate model. They iterate over the models to gradually improve them; therefore, after codifying, they will get back to the stakeholders until they have all the knowledge to fully model the information landscape. In the upcoming sections, we dive deeper into how this codification is performed.

5.2 Minimal Input Necessary for Constructing an ArchiMate Model

Little can be found in the literature on what precise input is necessary for the construction of an ArchiMate model. Literature speaks more of categories (e.g. company strategy) than of how exactly this category should be described (e.g. a certain strategy model). Chaabouni et al. (2019) made an ArchiMate model for their research. To model this, they needed the process requirements, the goal and strategy, the task processes in the workpackages, the workpackages, and the technological components [CBW+19]. Another research study states that the process requirements are necessary, as requirements are the most significant part of the development of an EA model [TLT14]. Moreover, research on the modelling of EA

using ArchiMate states that the core artefacts are a strategy, process, organisation, application, software and technical infrastructure specification, as well as a specification of the dependencies between the three layers [ZZ22]. Winter and Fischer (2006) name almost all of these categories as well, except the company strategy. However, they add that knowledge of the structure (hierarchy) is necessary as well [WF06].

As shown in Appendix B, we have not asked the Enterprise Architects about whether they agree with all these categories. We only asked them about the categories that we have found within at least two of the studies, as the researchers did not agree on all categories. Therefore, we only make use of the categories that were named twice. This way, the categories are scientifically well-backed. These categories are the following:

- Requirements of the process [CBW$^+$19][TLT14];
- Strategy of the company [CBW$^+$19][ZZ22];
- Specification of the technological components [CBW$^+$19][ZZ22][WF06];
- Specification of the organisation [ZZ22][WF06];
- Specification of the process steps [ZZ22][WF06];
- Specification of the applications [ZZ22][WF06];
- Specification of the software components [ZZ22][WF06];
- Specification of the dependencies between the three layers [ZZ22][WF06].

Input Category	**Occurrence in interviews**
Architect agrees with input found in literature	7
Strategy of the company	7
The technical components	7
The applications	7
The process steps	7
How the data is saved	4
The target audience	4
Process requirements	3
Documentation	3
Relations between the components	3
The scope of the process	3
The actors	2
Definitions of the components	1
User stories	1

Table 5.2: This Table shows the input categories mentioned within the interviews, with the occurrence of how many interviews these categories are mentioned in

Table 5.2 shows all the input categories that were mentioned during the interviews with the Enterprise Architects. The Enterprise Architects believed that these categories were necessary as minimal input. The first named category, 'Architect agrees with input found in literature', which entails that the Enterprise Architects agree with the input specified at the beginning of this section. As stated in Section 3.7.2, we take the categories with an occurrence rate of five or higher into account and use these categories as backing for the literature. As these categories are already within the list we specified from the literature, that list can be seen as the input categories that are minimally needed to be able to construct an ArchiMate model according to the literature and professionals.

5.3 Requirements of the Method and Tool

5.3.1 Requirements Elicited in the Interviews

Requirements	Occurrence in interviews	Requirement type
Input needs to be secured	9	Usefulness of the tool
You need to be able to talk to the tool	7	Usefulness of the tool
You need to be able to input artefacts	6	Usefulness of the tool
The model should not be too detailed	4	Way of codifying
The decisions of the tool should be retrievable	3	Usefulness of the tool
You need to be able to choose the viewpoint	3	Usefulness of the tool
The output is according to the ArchiMate syntax	2	Usefulness of the tool
Output is reusable	2	Usefulness of the tool
It needs to build the repository	1	Usefulness of the tool
It needs to find patterns within the data	1	Usefulness of the tool
The output of the tool can be shared	1	Usefulness of the tool
The output type matches the type of architect	1	Usefulness of the tool
It has a structured way of working	1	Way of codifying
The tool gives me predictability	1	Usefulness of the tool
The tool has construction rules	1	Way of codifying
The tool shows the relationship between elements	1	Usefulness of the tool

Table 5.3: This Table shows the requirements elicited within the interviews, with the occurrence of how many interviews these requirements are mentioned in

Outputted viewpoint	Occurrence in interviews
Process viewpoint	5
Application architecture view	5
A view that shows the three layers	4
Technology view	2
Capability Map viewpoint	1
Dependency viewpoint	1
Information structure viewpoint	1
Layered viewpoint	1
Motivation viewpoint	1
Plateau & Gap viewpoint	1
Portfolio view	1
Primary ArchiMate viewpoint	1
Stakeholder views	1

Table 5.4: This Table shows the required viewpoints elicited within the interviews, with the occurrence of how many interviews these viewpoints are mentioned in

Table 5.3 shows all the requirements that were elicited during the interviews with the Enterprise Architects. It shows the raw data, thus only the code given in the qualitative analysis, the occurrence rate and the type of requirement. The type 'Way of codifying' entails that this requirement is about how the codification should be performed and thus is a requirement on the codification method. The type 'Usefulness of the tool' entails that the requirement is about how the tool should work or what the output of the tool should look like. Furthermore, Table 5.4 shows what types of viewpoints the Enterprise Architects want to get out of the codification. This gives us a deeper understanding of what elements should be codified and thus of the codification method. In Subsection 5.3.3, we turn the codes from this subsection and the information described in Subsection 5.3.2 into a final set of requirements.

5.3.2 Survey Results

As stated in Section 3.2, we did a survey to find out how AI is used and experienced within the practice of Enterprise Architects at the moment. The full survey can be found in Appendix C.

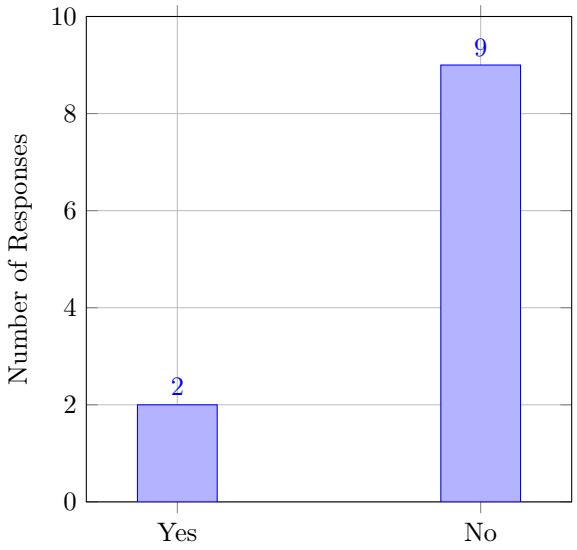


Figure 5.2: This figure shows how many respondents use AI within the codification process

Figure 5.2 shows that out of the eleven respondents to the survey, only two make use of AI within the process of codifying knowledge into ArchiMate models. One makes use of an AI agent to help him create an ArchiMate model. The other only uses AI for research purposes. They both have good experiences with using AI. One states that AI enhances their productivity and improves the consistency, quality and completeness of the model. The other gives a caveat, stating that you have to ask the right questions and know enough about the subject to tell apart the right information from the wrong. Because of AI, they have more capacity available for engaging with stakeholders, because the LLM performs the documentation of ArchiMate. Furthermore, with AI, they have a sparring partner that is always available. However, they see improvement points as well; they state that at the moment it is still a big challenge for an LLM to correctly codify knowledge from notes or reports into ArchiMate elements because those input types are inconsistent. Furthermore, they believe AI can't help them with cases that are new, secret or too complicated. Moreover, AI gets steered too much by what you say. Neither of these Enterprise Architects has tried to perform other tasks with AI within the codification process because of a lack of time or need.

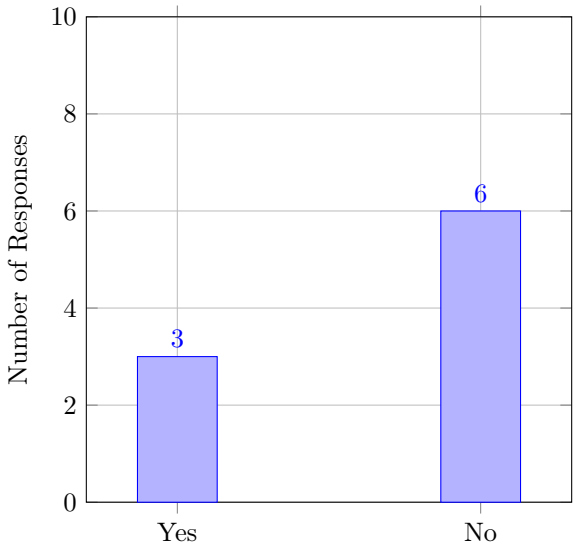


Figure 5.3: This figure shows how many respondents who do not use AI within the codification process have tried to use AI within the codification process

Figure 5.3 shows that from the nine respondents who do not use AI within the codification process,

only three have tried to perform tasks within this process with AI. They all tried to model architectures with the help of LLMs, but have mixed feelings about this experience. One sees usable results, whereas another states that the ArchiMate model wasn't of good quality. Furthermore, the text analysis went well, as well as the discovery of the problem, so that you can start faster. However, they see some improvement points as well. Direct translation to a readable ArchiMate file did not go well and LLMs weren't able to further explore and deepen the problem. The reasons for these problems were that they weren't allowed to use client-sensitive information within their prompts, resulting in generic answers.

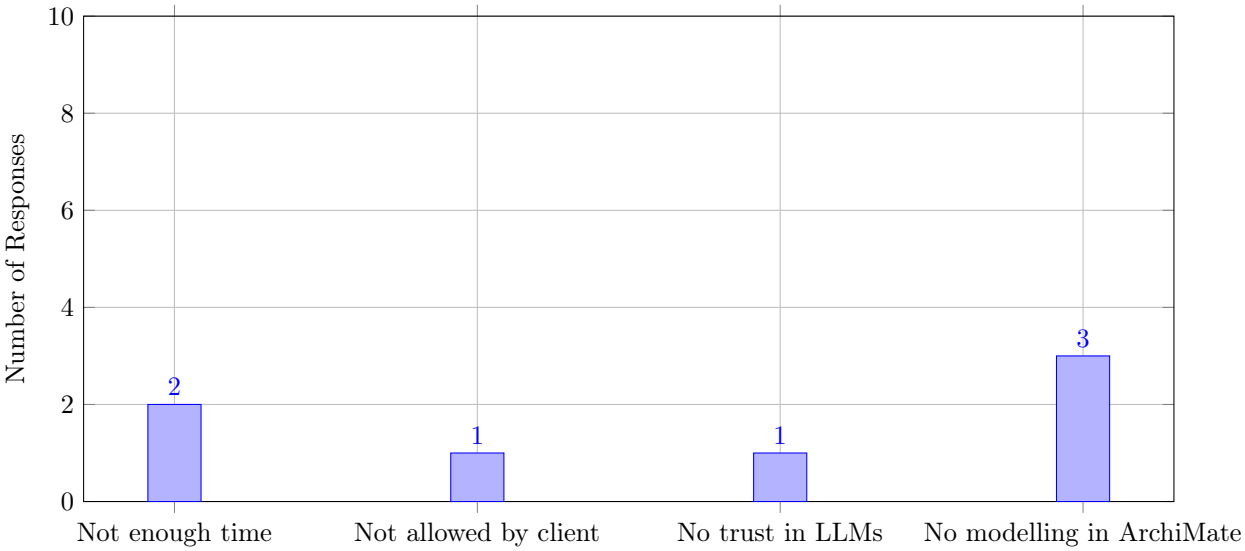


Figure 5.4: This figure shows why respondents do not make use of or have tried to work with AI

Figure 5.4 shows why the six respondents who do not use or have tried to work with AI made this decision. Most of these respondents stated that because they do not model in ArchiMate, they do not make use of AI in the process of codifying knowledge into an ArchiMate model. Some of them did not model in ArchiMate as they did not have a full understanding of the language yet. Others stated that they do not have enough time within their assignments to try to work with AI. Furthermore, a respondent does not work with AI as the client they work for does not allow them to use it. Therefore, they did not try to work with it. Lastly, a respondent shared a lack of trust in the capabilities of LLMs in codifying knowledge into an ArchiMate model.

5.3.3 Final Requirements

As stated in Section 1.3, we focus on steps B, C and D of the ADM cycle. Therefore, we are focused on the following viewpoints, which are the viewpoints with an occurrence rate of 2 or higher as well:

- Process viewpoint
- Application architecture view
- Technology view
- A view that shows the three layers (business, application, technology)

Combining these viewpoints, Section 5.3 and Section 5.3.2, gives us a final set of requirements for our designs. We distinguish between Functional Requirements (FR) and Non-functional Requirements (NFR). The requirements for the codification method are the following:

- FR-01: The codification method shall contain steps for the codification of the inputted knowledge according to the four predefined viewpoints: Process viewpoint, Application architecture view, Technology view and a view that shows the three layers;
- FR-02: The codification method shall only codify the active elements of the Business, Application and Technology layer so that the output is not too detailed;

- FR-03: The codification method shall provide a structured approach;
- FR-04: The codification method shall define a set of construction rules for applying codification;
- FR-05: The codification method shall codify unstructured and inconsistent input into a structured ArchiMate format.

FR-01 is based on the viewpoints that need to be outputted by the model and, therefore, based on the raw requirements from the interviews. FR-02, FR-03 and FR-04 are based on the requirements elicited in the interviews as well. FR-03 and FR-04 talk about a structured approach and a set of specific construction rules. These are not predefined and are developed in Chapter 6. FR-05 is based on the survey, as a respondent stated that it is a big challenge for LLMs to codify from inconsistent input from notes, which are mostly unstructured. Why in this case, the requirements with an occurrence rate of 1 out of 15 are still used as final requirements is explained in Section 3.7.2.

As stated in Section 3.7.2, for the tool, we take the requirements elicited in the interviews with an occurrence rate of three or higher and those that we found within the survey into account. Those from the survey do not need an occurrence rate of three or higher, as these requirements come directly from the question of why Enterprise Architects do not make use of AI within the process of codification. Therefore, taking away these reasons is important, as otherwise the method and tool are only usable on paper and not in the real-world. Those from the interviews do need this minimum occurrence rate to ensure that the requirements are not only wanted by one Enterprise Architect, but by at least three of the fifteen.

- NFR-01: The tool shall ensure that all input data is processed and stored in a secure manner to prevent unauthorised access;
- FR-06: The tool shall provide a chat interface that allows users to submit prompts;
- FR-07: The tool shall provide an interface for users to upload artefacts;
- FR-08: The tool shall store and allow retrieval of the rationale behind its decisions;
- FR-09: The tool shall allow users to select the viewpoint for which the output is generated;
- NFR-02: The tool shall reduce the time required to perform codification compared to the current process;
- NFR-03: The tool shall be easy to use for users with minimal training;
- NFR-04: The tool shall enable users with little knowledge of ArchiMate to perform codification tasks.

NFR-01 comes from the requirements elicited in the interviews and the problem that clients do not allow Enterprise Architects to make use of AI because of security issues. FR-06, FR-07, FR-08 and FR-09 are all requirements that come from the interviews. NFR-02 is based on the problem stated in Chapter 4 that the process of codification is very time-consuming. Furthermore, NFR-03 is based on the survey where respondents stated that they do not have enough time to start using AI. Therefore, ease of use and minimal training are important, as the users do not have the time to learn how to use a difficult tool. Lastly, NFR-04 is a requirement, as some respondents stated they do not model in ArchiMate yet, as they lack the knowledge. Therefore, a requirement for this tool is that it can be used with little knowledge as well.

Chapter 6

Design & Development

In this Chapter, we design the codification method and evaluation tool. We begin by constructing a method on how one could map retrieved information into an ArchiMate model using the information on modelling steps, minimal input and requirements found in Chapter 5. Furthermore, we make use of the description of the ArchiMate elements shown in Appendix A. After designing this codification method, we designed an evaluation tool that makes use of an LLM to follow the codification method so that we can evaluate whether this method works well and whether LLMs are able to follow these steps and come up with a good result.

6.1 Codification Method

When looking at the steps of the modelling process shown in Figure 5.1, the first steps of the codification method are shown. First, you have to set the scope of the information landscape that you are going to model. Within this step, you need to define which end-to-end process needs to be modelled in ArchiMate and thus between which boundaries the process lies. After setting the scope, the step within Figure 5.1 is to make a codification model to use within the conversations you are going to have. To be able to make a model for these conversations, you have to have a viewpoint that you are going to talk about. Therefore, the second codification step is to choose a viewpoint. In our codification method, the options for the viewpoints are defined in Section 5.3.3.

The third step in the modelling process is to interview the stakeholders. During these interviews, you need to gain certain knowledge; therefore, the third step within the codification method is to gain knowledge about the information landscape. As stated in Section 5.2, this means that you need to gain knowledge on the following categories:

- Requirements of the process;
- Strategy of the company;
- Specification of the technological components;
- Specification of the organisation;
- Specification of the process steps;
- Specification of the applications;
- Specification of the software components;
- Specification of the dependencies between the three layers.

The next step in the defined modelling process is to codify the found knowledge into an ArchiMate model. In Section 1.3, we set the scope to be on the B, C and D steps of the ADM. This means our focus lies on the modelling of ArchiMate elements of the business, application and technology layers. Furthermore, business capabilities and passive elements are not modelled within this study. The main elements in the business layer are Business Actor, Business Role, Business Function, Business Service, Business Process

and Business Event. The main elements in the application layer are Application Component, Application Function, Application Service and Application Interaction. The main elements in the technology layer are Technology Components, Technology Function, Technology Service and Technology Interaction [Vis25a]. The information on which we base these main elements is grey literature written by Visual Paradigm, which is one of the bigger parties within the Enterprise Architecture modelling applications. As this research has practical elements in it, this practitioner's view can be used, but these elements are thus not scientifically proven to be the main elements.

As Technology Interaction and Application Interaction add an additional layer of behavioural abstraction of the model that overlaps with processes, functions and services, we choose to leave these two elements out of our method. Furthermore, the Technology Components are Node, Device and System Software. Therefore, we codify these three elements within our method.

Artefacts	Interview 1	Interview 2	Interview 3
Codification method	The signals should be consequent, if X is used for an actor, do not use X for another element as well. Node is quite specific, put it as the lowest element in the table.	Try to find some more signals or conditions on when an element occurs. Look at the ArchiMate principles for ideas	Add 'is responsible for' and 'activity' to actor Make it clearer that the definitions used are retrieved from ArchiMate documentation
Evaluation tool	The download button should be higher, so you directly see it	The download button should be higher	Looks good
ArchiMate model	There are some services and relationships missing. A solution could be to add a function that returns all elements that miss relationships, so that the user can state which relationships need to be added. About 80% of the repository would be a good result.	80% of the repository is a good result. There are double relationships, where no doubles are allowed.	80% is a good result, try to find a better way to get a higher percentage of correct relationships as well

Table 6.1: This table shows the feedback of the respondents on the artefacts shown in the evaluation interview

As stated in Section 3.3, we performed an interview with an Enterprise Architect to find out which signals there are within documentation or interview transcripts. Based on this interview, a codification method of the elements was constructed. This table can be found in Appendix D. Within the interviews evaluating the method and tool, the respondents were shown this codification method. During the first sprint, we evaluated this method alongside the tool (frontend shown in Appendix E) and the ArchiMate model the tool created using the codification method and based on the inputted case of Capgemini, which is shown in Appendix F. The results of these evaluation interviews are shown in Table 6.1. How both these interviews are set up can be found in Section 3.7.1.

ArchiMate element	Formal definition according to Appendix A	Example	Hints to finding	Signalwords
Business Actor	Represents a business entity that is capable of performing behaviour	Customer	Is a person or organisation	A does B A is responsible for (A is actor)
Business Role	Represents the responsibility for performing specific behaviour, to which an actor can be assigned, or the part an actor plays in a particular action or event	Sales Representative	Findable in reference architectures Coupled to actors and process steps Function of an actor in the process step	A does B as C (C is role)
Business Process	Represents a sequence of business behaviours that achieves a specific result such as a defined set of products or business services	Fulfill order	Is a verb Is an activity	B is done (B is process)
Business Function	Represents a collection of business behaviour based on a chosen set of criteria (typically required business resources and/or competencies), closely aligned to an organization, but not necessarily explicitly governed by the organization	Sales Management	Is a capability a company has that is executed by a role with an actor Is a noun Typically needs business resources or competencies	Carries out Processes Executes
Business Service	Represents explicitly defined behaviour that a business role, business actor, or business collaboration exposes to its environment	Customer Support	Is a service Something that adds value Is a noun Is necessary to couple process steps and functions to actors Business services are for external actors, not for internal	D gets delivered D serves A/C (D is service)
Business Event	Represents a business-related state change	New order comes in	Because of a business event, a process begins to run Is a noun	E takes place E happens (E is the event)
Application Component	Represents an encapsulation of application functionality aligned to implementation structure, which is modular and replaceable	CRM System	Look at documentation of the supplier Is a noun If a system is mentioned, then it is an application component	System Bought Implemented in project
Application Function	Represents automated behaviour that can be performed by an application component	Customer Data Management	Is a functionality that can be performed by (a) component(s)	A functionality Parts of
Application Service	Represents an explicitly defined exposed application behaviour	Customer Data Retrieval	Is what the user needs the component for to perform the process step	User does F with Application Component Business Process or Business Function runs on F (F is application service)
Technology Service	Represents an explicitly defined exposed technology behaviour	Deployment	Is a noun Is the service a technology node\function delivers to an application component If node/device is a server, then the service is called Deployment	Delivers Services
Device	Represents a physical IT resource upon which system software and artifacts may be stored or deployed for execution	Serverfarm	Is a noun A node can consist of (a) device(s)	Consists of Is part of
System Software	Represents software that provides or contributes to an environment for storing, executing, and using software or data deployed within it	Oracle Application Server	Is a noun Is responsible for that the service can be delivered Is a collection of functions	Has the function Runs on
Technology Function	Represents a collection of technology behaviour that can be performed by a technology internal active structure element	Data Storage	Is a verb Is what the software executes	Is executed Software does G (G is the function) Engine
Node	Represents a computational or physical resource that hosts, manipulates, or interacts with other computational or physical resources	Database Server	Findable in documentation Is a noun	Cluster Server Compute node Virtualisation node

Table 6.2: The elements we codify complemented with its definition, an example, hints to finding it and signals

Table 6.2 shows the ArchiMate elements that we codify with their formal definition. This formal definition is retrieved from Appendix A. To give the Enterprise Architect more insight into what this element entails, we provide an example of what this element could be. These examples are from the same process so that the elements can be seen in relationship to each other as well. All examples, apart from those from the Node, Device and System Software, are retrieved from Visual Paradigm (2025) [Vis25a]. Based on an interview with an Enterprise Architect, we elicited hints for finding these ArchiMate elements. Furthermore, using this interview, we described which signals within documentation or interview transcriptions there are that can lead an Enterprise Architect to choose a certain ArchiMate element. This led to the table shown in Appendix D. We have updated this table after the evaluation interviews we had after the sprint. Using the outcomes of these interviews shown in Table 6.1, we improved the method even further, leading us to the results in Table 6.2.

Using Table 6.2, an Enterprise Architect can codify the retrieved information. However, this is not the only thing that should be done when creating an ArchiMate model. The elements are related to each other and therefore need to be connected via relationships. Furthermore, it is customary to model the business layer up top, the application layer in the middle and the technology layer underneath. These layers need to be connected via the Service elements.

From \ To	Business Event	Business Actor	Business Role	Business Process	Business Function	Business Service
Business Event	X	X	X	Trigger	X	X
Business Actor	X	Composition	Assignment	Assignment	Assignment	X
Business Role	X	X	X	Assignment	X	X
Business Process	X	X	X	Trigger	Assignment	Realization
Business Function	X	X	X	Assignment	X	Realization
Business Service	X	Serving	Serving	X	X	X
Application Component	X	X	X	X	X	X
Application Function	X	X	X	X	X	X
Application Service	X	X	X	Serving	Serving	X
Technology Node	X	X	X	X	X	X
Technology Service	X	X	X	X	X	X
Technology Device	X	X	X	X	X	X
Technology Software	X	X	X	X	X	X
Technology Function	X	X	X	X	X	X

Table 6.3: An overview of all the relationships between the elements our method codifies and the business layer elements

From \ To	Application Component	Application Function	Application Service
Business Event	X	X	X
Business Actor	X	X	X
Business Role	X	X	X
Business Process	X	X	X
Business Function	X	X	X
Business Service	X	X	X
Application Component	Flow	Assignment	Realization
Application Function	X	X	Realization
Application Service	Serving	Realization	X
Technology Node	X	X	X
Technology Service	Serving	Serving	X
Technology Device	X	X	X
Technology Software	X	X	X
Technology Function	X	X	X

Table 6.4: An overview of all the relationships between the elements our method codifies and the application layer elements

From \ To	Technology Node	Technology Service	Technology Device	Technology Software	Technology Function
Business Event	X	X	X	X	X
Business Actor	X	X	X	X	X
Business Role	X	X	X	X	X
Business Process	X	X	X	X	X
Business Function	X	X	X	X	X
Business Service	X	X	X	X	X
Application Component	X	X	X	X	X
Application Function	X	X	X	X	X
Application Service	X	X	X	X	X
Technology Node	Composition	Realization	Composition	Composition	Assignment
Technology Service	X	X	X	X	X
Technology Device	X	Realization	X	X	Assignment
Technology Software	X	Realization	X	X	X
Technology Function	X	Realization	X	X	X

Table 6.5: An overview of all the relationships between the elements our method codifies and the technology layer elements

Following the research by Perez-Castillo et al. (2021), we created tables similar to Figure 2.7 [PCCRP21]. Table 6.3 shows the possible relationships between all the ArchiMate elements we codify and the business layer elements we codify. Table 6.4 shows the possible relationships between all the ArchiMate elements

we codify and the application layer elements we codify. Table 6.5 shows the possible relationships between all the ArchiMate elements we codify and the technology layer elements we codify. In the first column, the tables show from which element the relationship goes and in the first row, the tables show to which element the relationship goes. If the table shows an X, it means there (normally) is no possible relationship between the two elements. Using these tables, the Enterprise Architect can model the relationships between the codified elements.

However, not all elements should be related. Therefore, the Enterprise Architect checks which elements are named together. For example, when the documentation states that a CRM System is deployed by a Database Server, an Enterprise Architect knows that the node Database Server realises the Technology Service Deployment, which serves the Application Component CRM System. Thus, before modelling the relationships, an Enterprise Architect goes over the found elements and checks within the tables whether there is a possible relationship between elements and then checks the documentation or interview transcriptions to find out whether this relationship actually exists.

The modelling is done based on the selected viewpoint. Therefore, when the process viewpoint is chosen, only the business layer elements need to be codified. When the application architecture viewpoint is chosen, only the application layer elements need to be codified. When the technology viewpoint is chosen, only the technology layer needs to be codified. Lastly, when the view that shows the three layers is chosen, all the elements we identified need to be codified.

Thus, the codification method consists of the following steps:

1. Set the scope;
2. Determine the viewpoint;
3. Gain knowledge on the information landscape;
4. Codify the discovered knowledge into elements using Table 6.2;
5. Check whether elements can have a relationship with each other using Tables 6.3, 6.4 and 6.5;
6. Check if the elements do have a relationship with each other;
7. Model the elements and their relationships.

6.2 Evaluation Tool

As this study is about the codification of unstructured knowledge of information landscapes into an ArchiMate model, we did not research multiple angles on how this tool should be designed. Within this section, we explain what design choices we made and why we made these exact choices. As described in Section 1.1 and Subsection 2.5.2, LLMs show the potential for performing codification tasks. To perform the codification steps set in Section 6.1, we chose to use AI agents within a Multi-Agent System (MAS).

6.2.1 AI Agents and a Multi-Agent System

AI agents are software programs that perform tasks (semi) autonomously. This way, these agents simulate human reasoning, learning and decision-making. AI agents can process large datasets, anticipate issues, match patterns and deliver results faster than humans. The benefits of using AI agents are, amongst other things, the enhancement of productivity, the reduction of costs, the improvement of service quality and the capacity to produce data-driven insights [NSDB25]. Multi-Agent Systems have come to light as a flexible and powerful example within AI. A MAS enables the solution of large-scale, dynamic and complex problems by decomposing these problems into manageable, smaller subtasks which are handled by multiple AI agents. Thus, within a MAS, agents do not operate in isolation, but collaborate and interact with each other. This collaborative behaviour enhances system performance significantly. This leads to a MAS exhibiting key characteristics such as adaptability, scalability, fault tolerance and robustness. Furthermore, as a MAS has a distributed nature, it reduces the level of dependency on a central controller. This enables the system to continue functioning even if some agents behave unexpectedly or fail [KFM+26].

Based on these characteristics, we chose to make use of a MAS. As we want our tool to solve the problem stated in Section 4 that codification is very time-consuming, the help of AI agents can make it more efficient while keeping the ability to reason and make decisions like humans. Using a MAS enhances these abilities and therefore can lead us to better results.

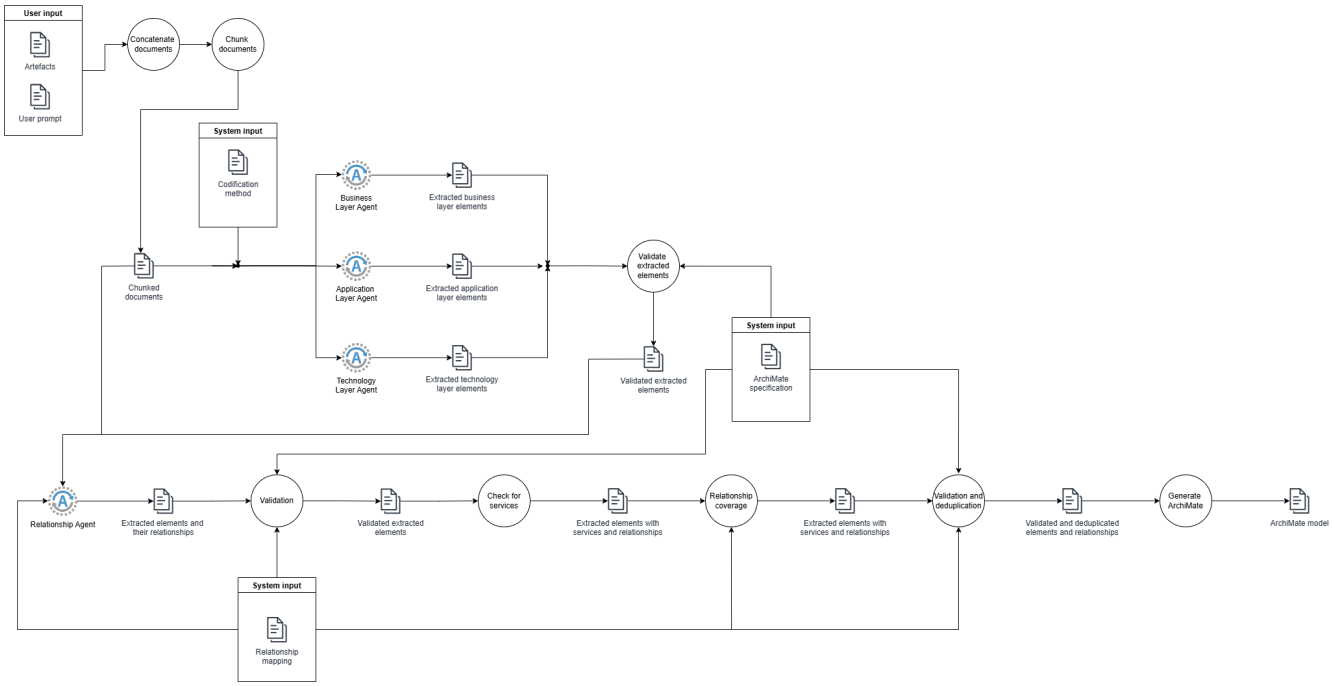


Figure 6.1: The workflow of the Multi-Agent System

Figure 6.1 shows the workflow of the MAS that we constructed. It starts with the user inputting the artefacts, which consist of the knowledge the user gathered on the information landscape and the user prompt in which the scope needs to be explained. The inputted documents are then concatenated and put into chunks which fit into the context window of the LLM. Within our system, an AI agent calls upon an LLM to perform a certain task. The context window means the amount of text the LLM can process at once [RLB+22]. The codification method of Table 6.2 and the chunked documents are then entered into three different AI agents that extract the ArchiMate elements per layer. We chose to use three AI agents for this task instead of one, as the performance of AI agents is significantly better when the tasks are smaller [ZYX+26]. After extracting all the elements, the elements are validated to check whether they conform to the ArchiMate specification shown in Appendix A.

The validated, extracted elements, chunked documents and Relationship mapping (according to Tables 6.3, 6.4 and 6.5) are then inputted into the Relationship Agent, which aims to find relationships between the extracted elements. These relationships are then validated based on the Relationship mapping and the ArchiMate specification, so that we are sure that every generated element and relationship is valid. As services are less indicated within the documents, but are necessary for the coupling of the three layers, there is an extra check to find out whether services are realised by the elements that can and if not, to add these services. This is done only for the Application and Technology layers, as these services are most necessary for connecting the layers. Then there is a deterministic check on the elements to find out whether they have all the relations that they need based on the Relationship mapping. Using this mapping, it can check what relations it can have and based on the semantics, it checks if there is a potential element to relate to. Then again, all the elements and relationships are validated based on the ArchiMate specification and the Relationship mapping. From this final set of elements, an ArchiMate model is generated. Following this method, only the four agents have to be instructed on how to perform their element or relationship extraction.

6.2.2 Prompt Engineering of AI Agents

The instruction of AI agents is done via prompts. The prompts of the agent itself are called a system prompt. A system prompt is a critical control surface that is used for the specification of the behaviour of LLMs in agent and chat settings. They may contain many different types of instructions and information and by default, the agent then follows all these instructions given within the system prompt [MLLW25]. As these instructions set the guardrails in which the agent works and how the agent operates, the setting up of these system prompts is of vital importance to the outcome of the MAS. Setting up these prompts is called prompt engineering.

The core concepts of prompt engineering are based on understanding the behaviour of language models, information retrieval theory and cognitive load management. The role of prompt engineering within LLM interaction extends beyond an input-output relationship. It is the primary interface between the intentions of a human and the comprehension of a machine. Well-engineered prompts significantly improve the quality of the output generated by AI [Vis25b]. There are different ways one can engineer a prompt. Firstly, you can use zero-shot (ZSL), one-shot (OSL) and few-shot learning (FSL). ZSL is aimed at recognising objects that are not part of the training [SWvdH+21]. Thus, meaning that in a ZSL prompt, no examples are given of what the output should look like. OSL makes use of one input and one corresponding outcome in the prompt [WZC+22]. This way, an LLM receives a single example of what should be done and how to react. FSL leverages example-based learning principles to enhance AI system performance using few training examples. Using FSL, AI agents understand patterns from examples and are enabled to adapt to new tasks [Vis25b]. Research has shown that incorporating supplementary data improves LLM outcomes and that, for example, GPT-4 performs the best using FSL [JSF23]. Therefore, we have chosen to make use of FSL in our system prompts.

Furthermore, research showed that a well-constructed prompt significantly improved the ability of an agent to generate context-appropriate actions, maintain consistent performance under variable conditions and interpret complex inputs. Moreover, a MAS demonstrates improved collaborative behaviour when the agents are guided by standardised prompt frameworks, which reduces ambiguity and enhances synergistic task execution [KBL+26]. Thus, when writing the prompts, we should use a standard prompt framework and make sure the prompt is well-constructed. Additionally, role-based prompting strategies address critical challenges in response consistency and bias mitigation [Vis25b].

Following these results, we constructed role-based prompts that follow a standard framework and make use of FSL. The prompts of the agents follow the following framework:

You are a ~**type of agent**~agent. Your task is to ~**task**~.

1. Role & Objective
    - This part sets the role and objective of the agent to make sure it is a role-based prompt
2. Output format
    - This part defines what the output of the agent should look like. This way, the output is exactly what is expected as input for the relationship agent or for the generation of the ArchiMate model
3. ArchiMate knowledge base or Relationship definitions
    - This is the direct translation per layer of Table 6.2 to a system prompt
    - Or this is extra explanation and signals for the finding of relationships
4. Extraction rules or Relationship rules
    - These are extra rules on, for example, deduplication of elements and relationships that the LLM has to take into account
5. Internal reasoning steps
    - These are the steps the LLM should take when looking through the input text.
6. Few-shot examples

- Within this part, the examples that can be used for FSL are shown to the LLM.

7. Final instruction
    - This part gives a final instruction, mostly stating that the LLM should analyse the input and explicitly follow the rules, schema and steps written within the system prompt

The system prompt for the Business Layer Agent can be found in Appendix G, the system prompt for the Application Layer Agent can be found in Appendix H, the system prompt for the Technology Layer Agent can be found in Appendix I and the system prompt for the Relationship Agent can be found in Appendix J.

The last setting we looked at is the temperature of the LLM. The lower the temperature, the more deterministic the response. When raising the temperature, the LLM gets more creative [Vis25b]. As we want our AI agents to follow the codification method as much as possible, we set the temperature at zero to be able to get the most deterministic responses. When evaluating the results tool, we make use of the LLMs Claude Haiku 4.5 from Anthropic and GPT 5.4 from OpenAI. We chose these models as they were both available to us within the Capgemini environment and both models were able to get documents as input within the Capgemini environment. Furthermore, both are state-of-the-art models developed by two of the biggest companies within the field of AI.

Chapter 7

Demonstration

In this chapter, we demonstrate how our codification method performs when executed by a MAS by making use of an anonymised case study provided to us. To do so, we show the codification results of multiple LLMs. Furthermore, we explain how a user can work with the evaluation tool.

7.1 Generated Views

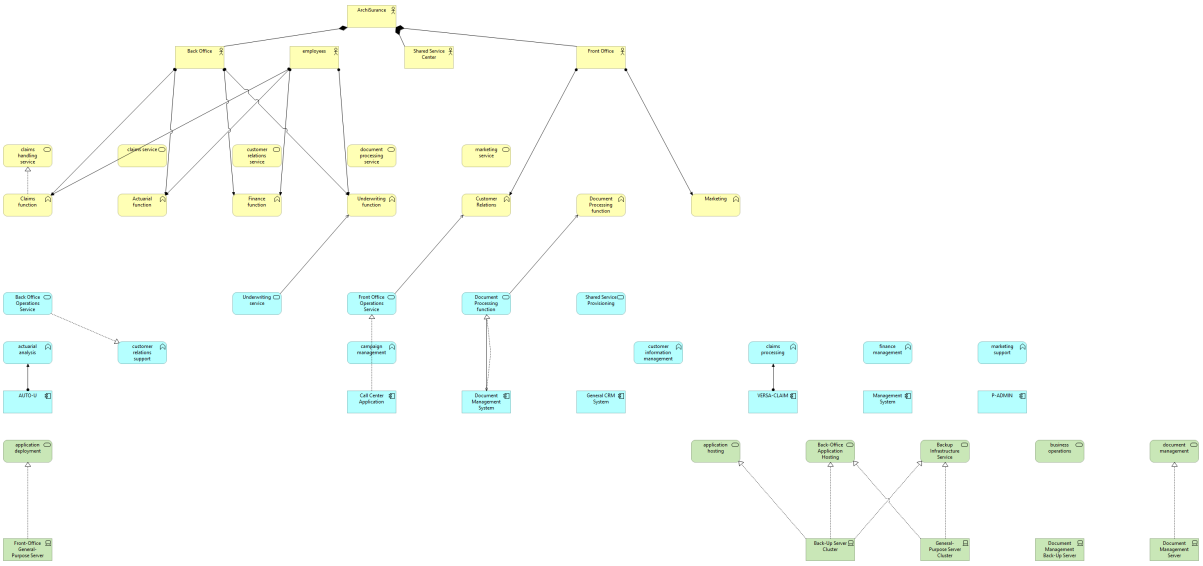


Figure 7.1: This view of the three layers is generated by Claude Haiku 4.5

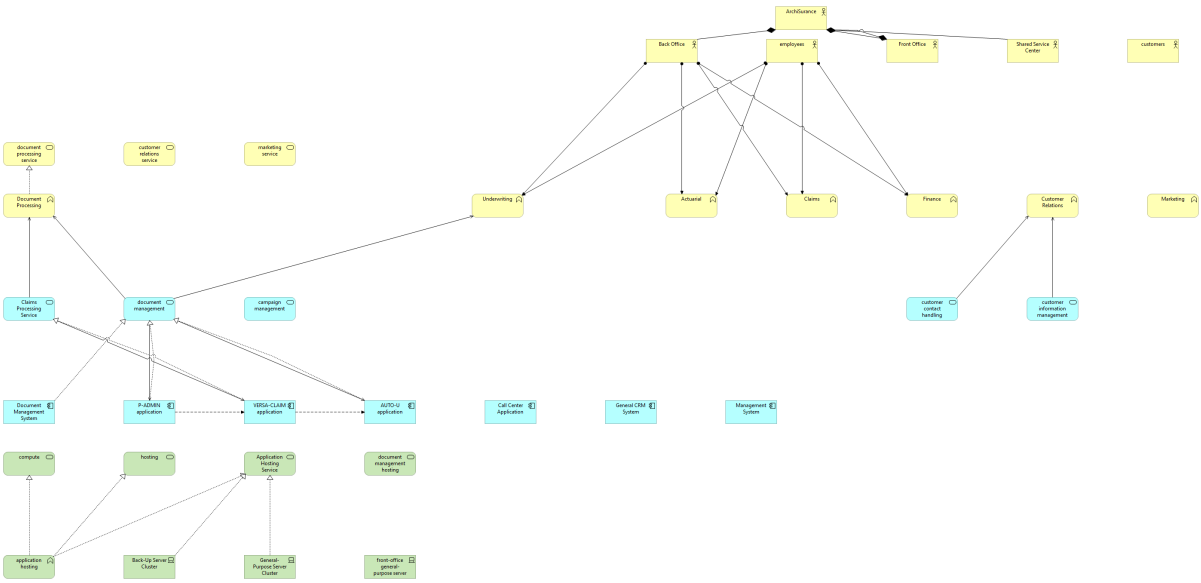


Figure 7.2: This view of the three layers is generated by GPT 5.4

Figures 7.1 and 7.2 show the views generated by the LLMs which follow the designed codification method. For a better look at these views, Figure 7.1 is shown in landscape in Appendix K and Figure 7.2 is shown in landscape in Appendix L. These views demonstrate that LLMs can generate an ArchiMate model via a MAS. The case study we used to perform this codification is shown in Appendix M.

7.2 How to Work with the Tool

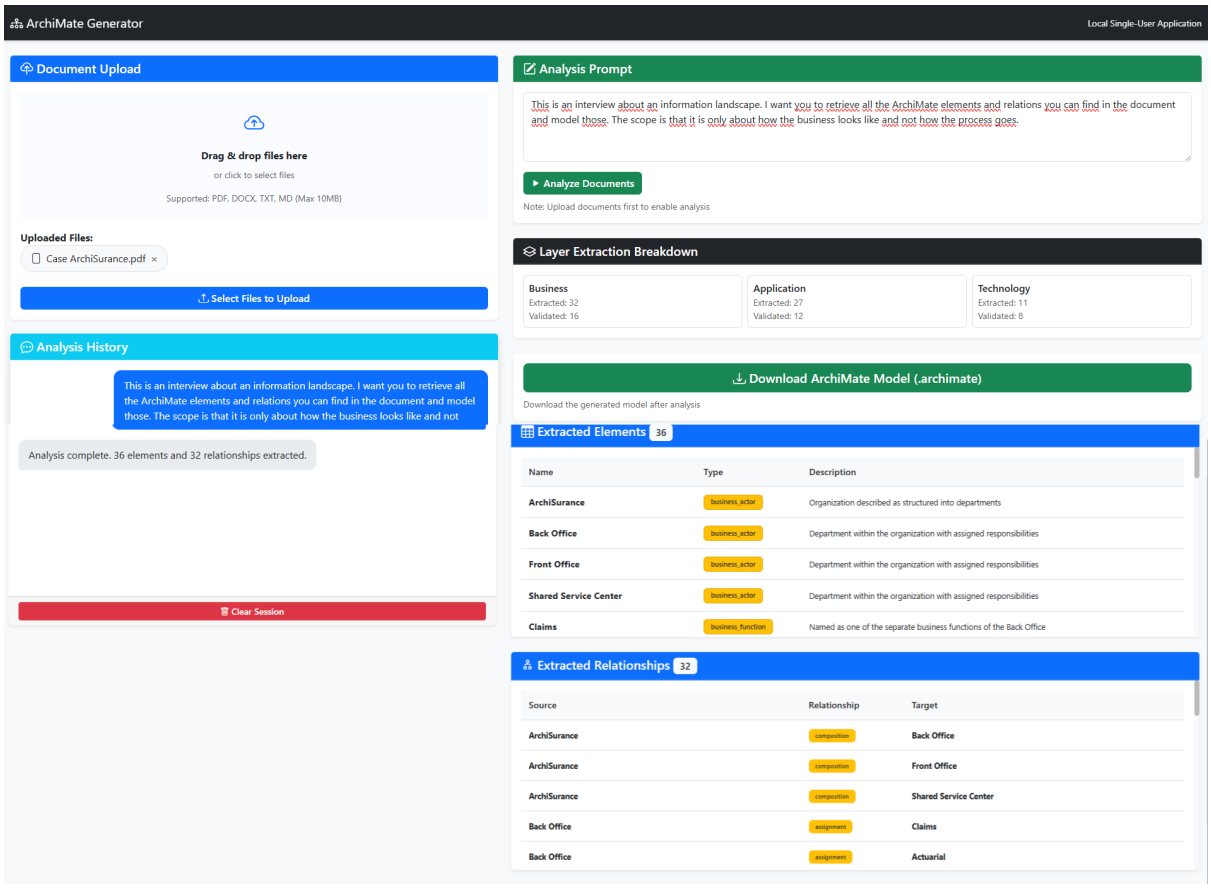


Figure 7.3: This figure shows the frontend of the tool

Working with this tool means following the codification method that we have designed. To show this, we explain it by showing the designed method combined with the steps you need to take in the tool. Underneath, the method is shown:

1. Set the scope;
2. Determine the viewpoint;
3. Gain knowledge on the information landscape;
4. Codify the discovered knowledge into elements using Table 6.2;
5. Check whether elements can have a relationship with each other using Tables 6.3, 6.4 and 6.5;
6. Check if the elements do have a relationship with each other;
7. Model the elements and their relationships.

A user sets the scope by making it explicit within the user prompt. This user prompt can be written in the top-right corner of the tool shown in Figure 7.3. Within this Figure, the user prompt we used is shown as well. Determining the viewpoint is not necessary for the user, as this tool models all the viewpoints we have determined in Section 5.3.3. After downloading, the user can choose to show these viewpoints within the Archi application, a tool for making ArchiMate models. The tool gains knowledge of the information landscape as the user uploads the knowledge he found via the upload button at the top-left. Then the codification can start by pressing the 'Analyze Documents' button underneath the user prompt. During the analysis, the MAS shown in Figure 6.1 starts performing the codification based on Table 6.2. Furthermore, it adds the relationships following Tables 6.3, 6.4 and 6.5. Then it models the elements and returns the user an overview of how many elements the tool extracted and how many of those were validated. Furthermore, it shows what elements were extracted with a description showing

the rationale. Moreover, the extracted relationships are shown. To see the model within the Archi app, a user has to download the ArchiMate model using the 'Download ArchiMate Model (.archimate)' button. This way, the tool follows all the codification steps we designed.

Chapter 8

Evaluation

In this chapter, we evaluate the codification method. First, we evaluate the precision, recall and time costs of the models generated by the LLMs compared to how the model would look and the time cost of modelling if one followed the codification method set in Section 6.1. These models and the codification method specified in Section 6.1 are evaluated in this section by an Enterprise Architect as well. Secondly, we look into the problems stated in Chapter 4 and evaluate whether our designs solve these problems. Lastly, we test whether the codification method and evaluation tool adhere to the requirements set in Section 5.3.3.

8.1 Evaluation Results

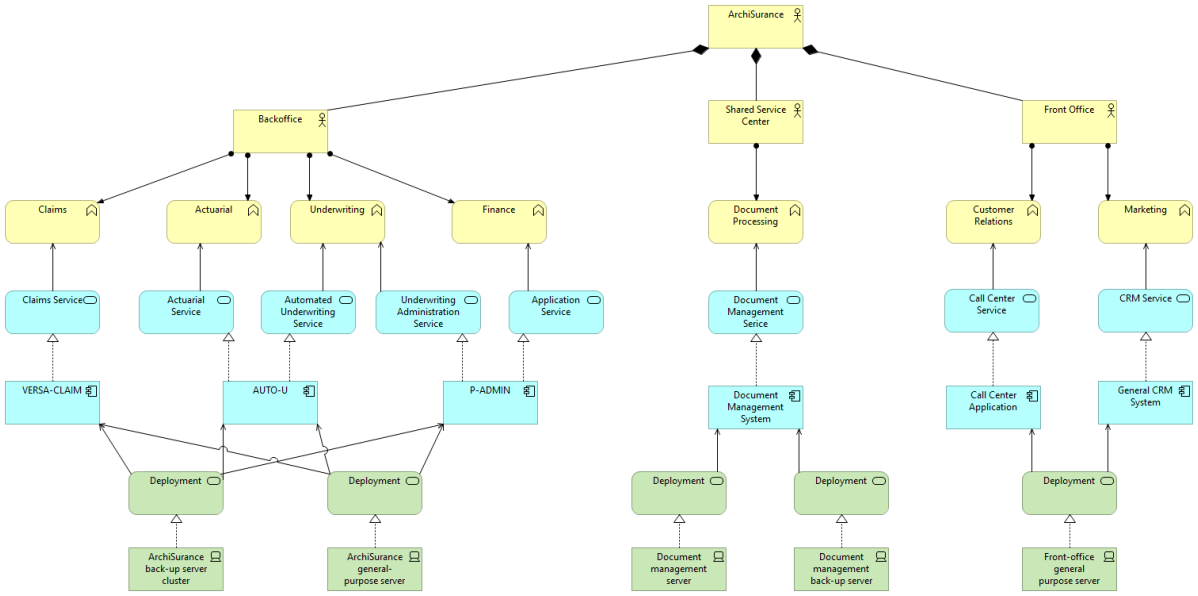


Figure 8.1: This figure shows the elements and relationships the model should codify

Metrics	Model used in Evaluation Interviews	Model generated by Claude Haiku 4.5	Model generated by GPT 5.4
Number of elements extracted	55 elements	47 elements	36 elements
Number of relationships extracted	38 relationships	30 relationships	32 relationships
Precision: Elements	31/55 elements = 0.56	32/47 elements = 0.68	27/36 elements = 0.75
Recall: Elements	31/35 elements = 0.89	32/35 elements = 0.91	27/35 elements = 0.77
Precision: Relationships	12/38 relationships = 0.32	18/30 relationships = 0.60	17/32 relationships = 0.53
Recall: Relationships	12/41 relationships = 0.29	18/41 relationships = 0.44	17/41 relationships = 0.42

Table 8.1: The evaluation metrics of the generated models

Figure 8.1 shows what the outcome of the case could look like if one followed the codification method set in Section 6.1. It shows a total of 35 ArchiMate elements and 41 relationships. Figures 7.1 and 7.2 show the results of the codification method, respectively performed by Claude Haiku 4.5 and GPT 5.4. On these models, we performed the evaluation process as described in Section 3.5. Table 8.1 shows the number of elements and relationships extracted and the precision and recall of the views made by the MAS using Claude Haiku 4.5 and GPT 5.4. Precision entails the fraction of the retrieved elements that are relevant compared to all identified elements [CHK24]. In our study, this means that the precision is the number of elements the LLM retrieved that can be found within Figure 8.1 divided by the total number of elements the LLM retrieved. The best possible precision rate is 1.0 because this means that the LLM only extracted elements that are correct and therefore hallucinated no elements. Thus, the closer to 1.0, the better. Recall is the fraction of the retrieved elements that are relevant compared to all possible relevant elements [CHK24]. In our study, this means that the recall is the number of elements that our LLM retrieved divided by the number of elements that can be found within Figure 8.1. The best possible recall rate is 1.0 because it means that the LLM found all the elements that are necessary for constructing the view. This means that with a score of 1.0, the full repository of the view is extracted.

The MAS we used in the evaluation interviews (Appendix F) has a very good score on recall. On the other hand, it still hallucinated a lot of elements that were not there, leading to a precision of 0.56. However, for the creation of that model, we used a very basic version of the case. The case used at that moment was just a list of what elements needed to be extracted (except for the services). Therefore, in the last sprint, we made use of a case that resembles the real-world cases more. This led to a more difficult case to analyse. Furthermore, after the evaluation interviews, we improved the codification method to the final method (Table 6.2). Thus, the scores on precision and recall of the two LLMs are on a harder case and with an improved codification method compared to those of the model we used in the evaluation interviews.

The final models that were generated made use of the improved case and method. This leads to the metrics shown in Table 8.1. The new method shows a clear improvement in precision, meaning that fewer elements are hallucinated. Furthermore, as the Enterprise Architects identified in Table 6.1, a score of 80% on found elements is very good. Claude Haiku 4.5 has a recall of 0.91 and GPT 5.4 a recall of 0.77. Meaning that one of the two is well above 80% and the other one is very close to 80%. What is good to notice is that we created the MAS working with Claude Haiku 4.5. Therefore, it makes sense that the recall of this LLM is higher than the recall of GPT 5.4, as during the development, we improved the method based on the model created by Claude Haiku 4.5. However, we do not believe that there is much overfitting, as GPT 5.4 still has a high score on recall and it extracts about as many relevant relationships as Claude Haiku 4.5 (17 compared to 18). Furthermore, the precision of GPT 5.4 is higher than the precision of Claude Haiku 4.5. The most improvement in sprint 3 is seen in the precision and recall of the relationships. It now finds fewer relationships, of which more are relevant. Therefore, less hallucination happens. Both models extract approximately the same number of relationships, of which about the same amount is relevant. In conclusion, both LLMs show quite good results on codifying the case into an ArchiMate model, as both create a repository with 77% to 91% of the total possible relevant elements and about 42% to 44% of the total possible relevant relationships.

Metrics	**Enterprise Architect**	**Claude Haiku 4.5**	**GPT 5.4**
Time costs	20 minutes and 26 seconds	2 minutes and 5 seconds	1 minute and 12 seconds
Costs	One-third of the hourly wage of an Enterprise Architect	$0,29	$0,24

Table 8.2: The costs in time and money of creating an ArchiMate model

Table 8.2 shows the time costs and costs in dollars of performing the codification following our designed codification method. It shows that not only do LLMs that follow our method get good results, but they also save time. We believe that the usage of LLMs saves money as well. However, as we do not know the exact hourly wage of an Enterprise Architect, we cannot certainly state this. We do believe that the hourly wage of an Enterprise Architect lies above $0,90 and thus the method saves money as well as time.

Artefact	Positive Points	Improvement Points
Codification Method	The elements are indeed commonly used elements It is a good method, Junior Enterprise Architects can use this as reference work or training method to learn how to find elements	Business en Data Object would be good additions
Evaluation Tool	Liked that it is really fast The tool is easy to work with	It would be nice if the tool could be connected to a current ArchiMate repository Standard user prompt templates would be useful
ArchiMate model	Liked the high scores of precision and recall of the elements.	Would be nice if every element also gets documentation added to it

Table 8.3: The final evaluation interview on the designs and outcome with an Enterprise Architect

Table 8.3 shows the results of the final evaluation we performed with an Enterprise Architect. As shown, he likes the codification method as it helps with finding commonly used elements and as it can be used by Junior Enterprise Architects as a reference work when they have to make an ArchiMate model. The improvement point of the method was out of scope for this study, but will definitely come back within future work. The practitioner was enthusiastic about how fast the tool generates a model that is very near the true model and believes this saves a lot of time. Furthermore, the ease of use is a positive point. The improvement points are things practitioners might add to the tool when putting it into practice; the same goes for the improvement points of the ArchiMate model. Lastly, the Enterprise Architect liked the precision and recall scores and believes that this means that in no time you get a good model, which you only need to complete.

8.2 Evaluation of Problem Resolution

The four main problems identified within the codification of knowledge to information landscapes were identified in Chapter 4 and are:

1. ArchiMate is open to interpretation;
2. It is difficult to codify knowledge to a model;
3. Codifying takes up much time;
4. It is difficult to keep codified models up-to-date.

Our research solves the first problem by introducing the codification method designed in Section 6.1. This method introduces a step-by-step way of performing the codification and resolves the interpretation problems by introducing Table 6.2. This table offers a method for determining which element to use when you encounter certain words or themes. When following this table, Enterprise Architects can interpret and codify ArchiMate elements in the same way. Furthermore, the designed codification method helps an Enterprise Architect to determine which element needs to be modelled based on the description, hints and signals described in Table 6.2. Thus making it easier to codify knowledge.

As shown in Table 8.2, making use of LLMs to perform our designed codification method saves the Enterprise Architect time, as the LLM performs the codification much faster while generating usable results. Therefore, our designed codification method can solve this problem when it is combined with a MAS. The last problem of the difficulty of keeping codified models up-to-date is not really solved by our method. At the moment, the codification method looks at documentation and then creates a new ArchiMate model. It cannot improve an already existing model, as our designed codification method does not take an already existing architecture into account when codifying the knowledge into elements. Therefore, this problem is not solved by our design. This is mentioned as well in Table 8.3 as an improvement point of the tool.

8.3 Evaluation of the Design Objectives

The following requirements for our codification method were set in Section 5.3.3:

- FR-01: The codification method shall contain steps for the codification of the inputted knowledge according to the four predefined viewpoints: Process viewpoint, Application architecture view, Technology view and a view that shows the three layers;
- FR-02: The codification method shall only codify the active elements of the Business, Application and Technology layer so that the output is not too detailed;
- FR-03: The codification method shall provide a structured approach;
- FR-04: The codification method shall define a set of construction rules for applying codification;
- FR-05: The codification method shall codify unstructured and inconsistent input into a structured ArchiMate format.

This codification method adheres to all these requirements. FR-01 is met as within the method, the viewpoint is determined and for all the elements within the possible viewpoints, a way of codifying is determined. Moreover, FR-02 is met as within our method, only the active elements are codified. Furthermore, FR-03 and FR-04 are met as our method provides a structured approach and sets construction rules within Table 6.2. Lastly, FR-05 is met as by using the signals and hints from Table 6.2, unstructured and inconsistent input can be codified into a structured ArchiMate format as shown in Figures 7.1 and 7.2.

The following requirements were set for our tool in Section 5.3.3:

- NFR-01: The tool shall ensure that all input data is processed and stored in a secure manner to prevent unauthorised access;
- FR-06: The tool shall provide a chat interface that allows users to submit prompts;
- FR-07: The tool shall provide an interface for users to upload artefacts;
- FR-08: The tool shall store and allow retrieval of the rationale behind its decisions;
- FR-09: The tool shall allow users to select the viewpoint for which the output is generated;
- NFR-02: The tool shall reduce the time required to perform codification compared to the current process;
- NFR-03: The tool shall be easy to use for users with minimal training;
- NFR-04: The tool shall enable users with little knowledge of ArchiMate to perform codification tasks.

The frontend of the tool is shown in Figure 7.3. When talking about parts or buttons of this tool, these can be found within this figure. NFR-01 is difficult to evaluate as the tool only runs on the device of the user at the moment. All input data is only temporarily stored within the tool and is cleared when the user clicks the red clear session button or when the user restarts the tool. Furthermore, the data never leaves the Capgemini environment as the API calls of the AI Agents are made within the Capgemini environment via a private connection. Therefore, NFR-01 is partially met, but when the tool is produced company-wide, extra safety precautions need to be set. FR-06 is met as there is a chat interface in which users can submit the user prompt. FR-07 is met, as there is a simple interface for uploading documents. FR-08 is met as the tool does this as well. For every retrieved element, you can see a description in the frontend showing its rationale. FR-09 is partially met. The user cannot choose the viewpoint on which the output gets generated, but all the viewpoints identified in Section 5.3.3 are shown in the ArchiMate tool Archi and there you can choose which view you want to see. NFR-02 is met, as Table 8.1 shows that the time it costs for the LLMs to codify the knowledge into an ArchiMate model is lower than the time it costs a human. NFR-03 is evaluated by the Enterprise Architect in the final evaluation interview, the results of which are shown in Table 8.3. These results indicate that the tool is easy to use and therefore this requirement is met. NFR-04 is not easily checked as all Enterprise Architects we have evaluated the tool with know the ArchiMate language. However, we believe the tool is easy to use and that you do not need any knowledge of ArchiMate to be able to upload a document, write a prompt, click on analyse and then on download file. Therefore, we believe this requirement is met, but we cannot prove it and therefore NFR-04 is not evaluated.

Chapter 9

Discussion

Within this chapter, we put the results evaluated in Section 8 into context. We start by interpreting the findings. Then we connect the findings to the existing literature. Lastly, we discuss the limitations of our research.

9.1 Interpretation of the Findings

Within this section, we take a look at our results from multiple viewpoints. Firstly, we start by discussing the topic that codification is more than direct translation. Secondly, we discuss the importance of explicit mapping rules for this codification. Thirdly, we discuss how the formalisation of codification can enable automation. Lastly, we discuss whether our evaluation tool validates the designed codification method.

9.1.1 Codification is more than Translation

Within Section 1.1, we described codification as the process of converting knowledge into information that is processable by machines and humans [JMHGH23]. This definition may suggest that codification primarily concerns the translation of knowledge. Thus, translating knowledge directly into a type of information. Existing approaches like ArchiRev make use of structured input and deterministic mapping [PCCRP21]. However, our findings revealed that in the case of codifying unstructured knowledge into an ArchiMate model, such as stakeholder interviews, codification is more complex. Figure 5.1 shows that Enterprise Architects need to take many steps to perform the codification. Thus, codification consists of three activities. Namely, the elicitation of information (via interviews or workshops), the interpretation of information (making a model and iterating over it) and then formally representing the information. The final step primarily concerns the formal translation.

The process shown in Figure 5.1 indicates that before formally modelling, architects reduce ambiguity by setting the scope. Furthermore, they go on to incorporate what they found and validate this through interaction with stakeholders. This shows that the modelling process is not a direct mapping exercise, but an iterative activity aimed at constructing a complete understanding of the information landscape. We have specified in Section 5.2 which information is necessary to be able to construct an ArchiMate model. However, even though the information is necessary, it does not mean that you can always gain it. As shown in the case study (Appendix M), the modelling of architectures involves incomplete information, as not all minimal knowledge was present, which leads to uncertainty and the making of assumptions, which was also mentioned by an Enterprise Architect as shown in Table 5.1. These findings suggest that the main challenge in the codification of EA knowledge is not in the formal representation of the architecture within ArchiMate, but in the construction of an understanding based on incomplete, fragmented and sometimes even conflicting stakeholder information. Therefore, codification should be seen as a process of inference and sense-making instead of a direct translation task.

These findings have implications for the building of a tool that performs the codification. As codification is not merely translation, such a tool cannot simply be a translator. This finding had a direct influence on the design of the artefact. Rather than designing an automated translation mechanism,

the design needed to support the full codification process by structuring reasoning, interpretation and formal modelling. Therefore, we designed a codification method. Our codification method supports the interpretation of information by introducing Table 6.2. Furthermore, Tables 6.3, 6.4 and 6.5 support inferring elements and relations by showing possible relationships. Rather than the prescription of deterministic mappings, our codification method structures the process by defining how elements can be elicited, interpreted and connected. This enables Enterprise Architects to infer elements despite incomplete information. For example, the elements Device and Application Component are identified and the Application Component seems to run on the Device. Our codification method then guides the Enterprise Architect to infer that there needs to be a Technology Service between these elements to connect them. This is because, following the relationship tables, it is required to connect these elements via a Technology Service. In this way, missing elements can be inferred when not explicitly stated in the information. Furthermore, as in the codification method, the first steps are to set the scope and define a viewpoint ambiguity can be diminished.

Thus, our findings indicate that the codification of EA knowledge is a process of interpreting knowledge and not just a simple translation of information. Before one can formally map knowledge to ArchiMate models, one needs to iteratively elicit, validate, and interpret knowledge that can be insufficient and sometimes even may be conflicting. These findings may explain why deterministic approaches such as ArchiRev perform well on codifying structured information, but are less suitable for unstructured knowledge. Consequently, support for EA codification should focus on assisting interpretation and inference of elements. Therefore, our codification method supports inference and interpretation and is not focused merely on the generation of models.

The results of the evaluation further support this interpretation. The codification method achieved good scores for recall on the identification of ArchiMate elements. However, the performance decreased for the modelling of relationships. This suggests that the recognition of explicitly stated concepts is easier than inferring how these elements relate to one another within the EA. Thus, the challenge in codification seems to lie in interpreting incomplete information and, based on that information, inferring the context that is required to establish the relationships. The modelling of relationships is exactly where Enterprise Architects need to be able to reason, because they need to be inferred. This could be a reason why the scores of precision and recall of the relationships are low. This further proves that the codification of unstructured information into an ArchiMate model is an inferential process and not just a direct translation.

9.1.2 The Role of Explicit Mapping Rules in EA Codification

One of the challenges identified in Chapter 4 was that the ArchiMate language is open to interpretation. This resulted in Enterprise Architects constructing different elements and models based on the same information. This finding was consistent with the literature, as research showed that this interpretation problem leads to different usage of the language between users [CBO11]. Therefore, the challenge with codification is not only one of selecting the appropriate ArchiMate elements, but also one of ensuring that modelling decisions are made in a consistent and transparent way by the Enterprise Architects.

Research has shown that the usage of deterministic mapping rules can codify structured input [PCCRP21] [KKPK23]. Our findings suggest that explicit mapping rules can be more useful when codifying unstructured information than the deterministic mapping rules described previously. Instead of only prescribing which element to identify, our rules structure the reasoning process through which Enterprise Architects can interpret unstructured information. The rules specify the elements that can be found, which narrows the scope and thus the ambiguity. Furthermore, by specifying the formal definition, an example, hints to finding the elements and signalwords, one can identify ArchiMate elements within unstructured information. This leads to the modelling choices that previously were tacit can now be made explicit.

Within Section 7, we demonstrated that the codification method could be operationalised by multiple LLMs. Then, in Section 8, we showed that the generated ArchiMate models are usable and reduce modelling time. These findings suggest that our codification method makes the codification process structured enough to support automation. The mapping rules therefore enable the connection of the expertise of Enterprise Architects to AI-supported codification. We did this by formalising the reasoning through mapping rules. Our findings suggest that formalising the reasoning through mapping rules

makes it possible to execute the codification process within a MAS. This way, the mapping rules do not fully eliminate interpretation, but they structure and constrain the process of interpretation, which leads to more reproducible and consistent EA codification.

These findings have broader implications. As modelling choices can be made explicit by following our codification method, the modelling process becomes more explainable as each modelling decision can be traced back to the formal definition, hints and signalwords that are defined within the codification method. This may reduce the inconsistencies that arise at the moment from interpretation. Furthermore, when using our codification method for training purposes, junior Enterprise Architects can learn a consistent approach to codification. This helps them as this method gives them a structured way of working and helps them understand the modelling choices of other Enterprise Architects better. Moreover, when incorporating our codification method, organisations can become less dependent on tacit knowledge and experience. As the usage of the method makes the modelling choices more explainable and explicit, an Enterprise Architect who did not make the model could be able to more easily understand why elements are modelled and therefore work with the ArchiMate model.

9.1.3 The Formalisation of Codification enables Automation

As argued in Subsection 9.1.2, the codification method formalises the reasoning process of the Enterprise Architects. The evaluation suggests that this formalisation provides the required structure that enables LLMs to incorporate the codification process as well. The success of the prototype cannot be attributed only to the capabilities of the LLM. The LLM never acts independently throughout the codification process, but the LLM performs the modelling task, which may be because of the structure the codification method provides. As the codification method breaks down the modelling task into a list of reasoning steps, the amount of implicit reasoning the LLM has to do is reduced. Instead, the codification method enables the LLM to go through a structured decision-making process.

Although the generated models have promising levels of precision and recall, the evaluation also showed that the LLM has difficulties with modelling relationships and preventing hallucinated elements and relationships. This observation suggests that the identification of elements is less demanding than the inference of the relationships between those elements. Elements are often described in the stakeholder information, whereas relationships require more reasoning, making them more difficult to automate. Thus, LLMs reduce the modelling effort for Enterprise Architects, but they do not eliminate the need for judgement. LLMs can accelerate codification, but Enterprise Architects remain necessary for the process as they need to complete the model. Thus, LLMs can augment Enterprise Architects and not replace them by supporting them during the construction of ArchiMate models. Section 8.1 shows that LLMs are effective, but not because they fully understand EA, but because they can implement a structured codification method that captures the reasoning of Enterprise Architects. Thus, the evaluation shows that LLMs do not automate the expertise of Enterprise Architects, but they automate a formalised representation of that expertise. These findings suggest that successfully applying LLMs to the modelling of EA may depend less on the capabilities of the LLM. It may depend more on the extent to which the reasoning of an expert can be formalised into a method. LLMs therefore may be more effective when the task it has to perform is formalised. Therefore, our codification method may be the enabler of automation, whereas the LLM then is the executor of the codification method.

These findings have implications that go beyond EA. The findings suggest that LLMs may be particularly suitable for supporting tasks that are knowledge-intensive when the reasoning of experts can be broken down into an explicit reasoning process. LLMs do not replace the experts who perform these tasks, but seem to be valuable in incorporating structured knowledge and reducing time for repetitive activities such as modelling.

9.1.4 Validation of the Codification Method

Section 8 indicates that the codification method and evaluation tool address three of the four challenges that were identified within Chapter 4. The designs formalise the underlying reasoning process and therefore support the codification of unstructured information and reduce both modelling effort and ambiguity. However, the evaluation shows that maintaining existing ArchiMate models is not within the scope of our codification method and tool. These findings show that the codification method addresses

the challenges associated with the codification of EA in Chapter 4.

More importantly, the evaluation demonstrates that the tool validates the practical use of the designed codification method. The successful implementation of the codification method by multiple LLMs shows that the codification method is explicit enough and structured in such a way that it can be executed by a MAS. Even though this method and tool are developed for ArchiMate, the underlying principle of formalising the reasoning of experts into codification rules may also apply to other CM languages such as BPMN or UML, if comparable mapping rules can be established for those languages. Therefore, the designs should not only be seen as a codification method executed by a MAS, but more as a proof of concept that the codification of EA can be formalised into an explicit process and then successfully executed by AI. In doing so, our design not only validates the technical possibility of AI-supported modelling of EA, but more fundamentally, the design validates the feasibility of formalising the reasoning process of EA codification by an expert.

9.2 Comparison with Existing Literature

Within this section, we compare our findings to the existing literature. We look into the scientific gap we discovered within Section 9.3. To do so, we look into what previous work concluded, what we discovered and how our findings support, extend or contradict previous work. We start by discussing our findings on the modelling of EA. Then we discuss the codification of EA. Lastly, we dive into LLM-supported modelling.

9.2.1 Modelling of Enterprise Architecture

Existing research describes EA modelling as a time-intensive task [FPB+12] [FBH+13] [TSC14]. Furthermore, modelling in the ArchiMate language is open to interpretation, leading to modelling decisions that differ between Enterprise Architects [CBO11]. Modelling decisions are now based on expertise, as a lot of the information that is used for EA modelling is unstructured, as the information is mostly from stakeholder interviews [FPB+12].

Our findings extend this understanding by indicating that the modelling of EA is not only a modelling activity, but it is an activity that follows a reasoning process. Instead of only making modelling decisions, Enterprise Architects define the scope, gain information, interpret this information, infer missing elements and then construct a model. These findings suggest that the differences between Enterprise Architects may not be based on modelling preferences, but on differences in reasoning. Therefore, improving EA modelling should not only focus on standardising ArchiMate but also on making the underlying reasoning process clear.

This decomposition of the modelling of EA suggests that the modelling consists of a series of steps and not of a single modelling task. Therefore, modelling does not only consist of knowledge of the ArchiMate language, but of the ability of an Enterprise Architect to resolve ambiguities, interpret unstructured information and construct an information landscape model. This would explain why modelling is still difficult despite the presence of a modelling language.

Our findings suggest that improving EA modelling is not only a matter of defining additional modelling conventions or extending modelling languages. Alternatively, comparable improvements may be realised by making the reasoning process reproducible and explicit. Therefore, this research contributes to the literature on EA modelling by demonstrating that the reasoning behind the EA modelling process can be formalised.

9.2.2 Codification of Enterprise Architecture

Knowledge is often tacit, making it difficult to transfer between persons via speaking or writing [CBB17]. To manage this knowledge, there are two options. Namely, personalisation and codification [LN15]. Existing research describes codification as the conversion of knowledge into information that is processable by machines and humans [JMHGH23]. Thus, codification transforms knowledge into explicit information. Within EA, approaches such as ArchiRev perform this codification via deterministic mappings. These mappings are focused on codifying structured information into ArchiMate models [PCCRP21].

Thus, existing research uses mapping rules to codify structured information.

Our findings extend these approaches by demonstrating that the codification of unstructured information cannot solely be understood as a direct translation task. Our findings indicate that codification requires an iterative process of interpretation, inference and validation before the knowledge can be codified. Thus, this research broadens the understanding of EA codification by indicating that successful codification of unstructured information requires formalising expert reasoning in addition to mapping rules.

By extending codification from deterministic mapping towards structured reasoning, our study extends the applicability of codification methods towards unstructured information. While the literature on knowledge management defines codification as the transformation of tacit knowledge into explicit information, our findings demonstrate that, within the context of EA, this transformation requires inference and structured interpretation. Whereas previous research depended on information that was already formally represented, our codification method enables the codification of unstructured information that often can be incomplete, context-dependent and ambiguous. These findings indicate that the successful codification of EA is not only dependent on mapping rules but also on supporting the reasoning process through which the codification method can be applied. Instead of replacing deterministic mapping approaches, our research extends them by extending EA codification from structured information towards unstructured information through formalising expert reasoning.

9.2.3 LLM-Supported Modelling

Research has shown that LLMs can understand unstructured data [TS26]. It is already possible to codify textual input into process steps using LLMs. Research shows that LLMs can understand textual input and turn this input into process steps using a prompt with clear instructions on how to break up an activity into multiple steps [dMCF25]. Furthermore, textual input can be codified into a process diagram [ZLCB24]. Lastly, a tool has already been created which allows its users to describe a workflow in natural language and then codify this natural language into a formal process model using LLMs [KBSvdA24b]. These studies highlight the potential of LLMs to automate CM tasks.

Our findings extend these studies by demonstrating that LLMs do not only need to depend on their own reasoning capabilities. In contrast, our designed codification method provides guidance that structures the modelling process before the generation of the model takes place. This way, the codification method provides reasoning steps and mapping rules to the LLMs instead of only making the LLM dependent on its own reasoning capability. These findings suggest that the application of LLMs to EA modelling is more dependent on the extent to which the expertise of a domain can be formalised into a process. Therefore, LLMs are effective when they can operationalise expert knowledge instead of replacing expert judgement.

Our research therefore contributes to the literature on LLM-supported CM by demonstrating that the combination of a codification method and an LLM is an operable way of model generation. Instead of only relying on the reasoning capabilities of an LLM, our proposed approach combines formalising domain knowledge with AI to support EA modelling. Thus, treating the LLM as an executor of a defined codification method.

9.3 Limitations

Within this section, we discuss the limitations of our research. We start with the generalisability of the codification method. Subsequently, we look into the scope of our codification method. Following this, we dive deeper into the limitations of our designs. Lastly, we comment on the limitations of the evaluation.

9.3.1 Generalisability of the Codification Method

The codification method is derived and evaluated within Capgemini. EA practices may differ between consultancies and organisations. This means that the identified process may not fully represent modelling practices in other contexts. As ArchiMate is open to interpretation, Enterprise Architects in other organisations may make use of other modelling conventions or reasoning strategies. This could result in different modelling decisions, indicators or mappings than those that we have incorporated within our

codification method.

Furthermore, during the study, we interviewed a limited number of Enterprise Architects (15) to gain knowledge on how the process goes, what knowledge is necessary and what requirements the codification method and tool should meet. All these Enterprise Architects work for Capgemini and have professional experience, but a larger and more diverse group of participants could have revealed additional modelling strategies that were not identified within this research. Moreover, for the designing and evaluation of the codification method and tool, only five evaluation interviews were performed, again all with Enterprise Architects working at Capgemini. Therefore, evaluation with a larger and more diverse group of participants could have revealed additional improvements of the codification method.

While the interviews enabled us to formalise much of the reasoning process of an Enterprise Architect, EA modelling is an activity that relies partly on tacit knowledge. Therefore, interview participants may have been unable to fully explain every modelling decision they make in practice. Hence, the proposed codification method only represents the reasoning that we could elicit from the Enterprise Architects and does not necessarily capture the full reasoning process when modelling EA. As a result, our codification method should be seen as a formalisation of the expert reasoning we elicited and not as a definitive representation of the modelling process of EA.

9.3.2 Scope of the Codification Method

In Section 1.3, we set the scope for this research. The scope was set on the B, C and D of the ADM cycle and thus on the Business, Application and Technology layer. Furthermore, the study does not take passive elements and business capabilities into account. Moreover, following the interview results, we selected four viewpoints in Section 5.3.3. However, by doing so, it remains uncertain whether our codification method can also be applied to the remaining ArchiMate layers, element types and viewpoints. Because of that, the effectiveness of the codification method cannot automatically be expanded to the viewpoints and ArchiMate elements that were outside the selected scope.

Furthermore, the tool can only be used for creating a new EA model. Even though this study demonstrates that EA modelling is an iterative process, with the tool, an Enterprise Architect cannot maintain or update existing models, but only perform the initial codification. Therefore, neither our codification method nor our tool is evaluated on maintaining or updating existing EA models. Because of that, within the evaluation, the applicability of the codification method is only demonstrated to be a support for the initial construction of EA models.

These design decisions were made to make sure the research scope was manageable and that a focused evaluation could be performed. Nonetheless, this implies that our findings should be seen as evidence for the applicability of our codification method within the specified research scope and during the initial construction of EA models. Therefore, the findings should not be interpreted as a demonstration of the applicability of our codification method to the full iterative ADM-cycle or the complete ArchiMate language.

9.3.3 Design Limitations

Even though our study demonstrates that the codification method can formalise the reasoning of Enterprise Architects, the codification depends on manually elicited expert knowledge on EA modelling via interviews. From these interviews, we derived mapping rules, such as hints and signalwords as well as the relationship tables. However, our codification method only captures the reasoning that we could elicit from the interviews. It is still possible that other modelling situations require mapping rules or reasoning approaches that we have not identified during our research. Furthermore, our codification method cannot simply be transferred to another modelling language such as BPMN or UML, as for these languages, our knowledge elicitation process needs to be repeated. Hence, adapting our method to another modelling language requires a comparable elicitation of expert knowledge and formalisation of the modelling process.

Furthermore, the tool works with static input. In contrast, our study has shown that EA is an iterative process. Enterprise Architects ask questions, try to clarify ambiguity and validate assumptions. However, our tool is not able to perform these tasks. Therefore, the tool is not able to resolve ambiguities

that require extra interaction with stakeholders. Hence, the codification method and tool should be seen as a decision-support system for Enterprise Architects and not as a fully automated modelling system.

9.3.4 Evaluation Limitations

For performing the evaluation, we made use of a training case of Capgemini. This case is on the information landscape of an insurance company. Therefore, our codification method is only evaluated on one case within one industry and with one type of documentation. Moreover, the evaluation was performed on a training case instead of a real-world EA project of Capgemini. Hence, applying our codification method to other (real-world) cases about other industries, types of documentation, or more ambiguous documentation may produce different results than those within our research.

Furthermore, the evaluation tool consists of several components: the codification method, the multi-agent architecture, prompt-engineered prompts and the LLM. Within our evaluation, we evaluate the model generated by the tool. Therefore, we evaluate the complete tool and not the contribution of each isolated component. Even though our codification method is positively evaluated by Enterprise Architects, our evaluation cannot determine to what extent the performance can be attributed to the codification method, the prompt-engineering, the LLM, the MAS or the interaction between these components. Moreover, as our study is about the codification of EA knowledge, we did not fully research all possibilities for constructing the tool. Therefore, there may have been better ways of constructing the tool that were not investigated within this study.

In addition to this, the codification method is evaluated using Claude Haiku 4.5 and GPT 5.4. Even if the method remains unchanged, another model could lead to other scores on precision and recall. Therefore, part of our evaluation reflects the capabilities of the used LLMs instead of only the codification method. Even though the precision and recall are calculated based on a reference ArchiMate model, EA modelling remains interpretative. Other modelling decisions may also be valid while leading to different scores on precision and recall. Hence, the evaluated precision and recall should therefore be seen as measures of agreement with the selected reference model and not as an absolute measure of correctness. Therefore, the performance should be seen within the context of how the evaluation was structured, including both the reference model and the LLMs.

Chapter 10

Conclusions

Within this chapter, we conclude our research. Firstly, we answer the research question and its sub-questions. Secondly, we dive deeper into what our research contributed to both the academic and practical fields of codification of EA. Lastly, we indicate what opportunities there are for future work.

10.1 Answers to the Research Questions

This study researched how the codification of unstructured information about an information landscape into ArchiMate models can be formalised and whether this formalisation can then be operationalised by LLMs. To do so, we proposed the following research question:

RQ *How can Enterprise Architecture knowledge be codified into ArchiMate models, and how can large language models support this codification?*

To answer this research question, we performed a DSRM study. This resulted in a codification method and the development of a MAS, which was used to evaluate the designed codification method. Within this study, we set the following sub-questions:

SQ1 What challenges with the codification of Enterprise Architecture knowledge exist?

SQ2 What are the mapping rules to codify Enterprise Architecture knowledge?

SQ3 What requirements do Enterprise Architects have for a tool that translates information landscape knowledge into ArchiMate models?

The answer to SQ1 can be found in Section 4, the answer to SQ2 can be found in Section 6.1 and the answer to SQ3 can be found in Section 5.3.3. Together, the answers to these sub-questions provide the basis for answering the research question.

Thus, to answer the research question, the results of our research demonstrate that unstructured EA knowledge can be codified into an ArchiMate model by formalising the reasoning process of Enterprise Architects into a codification method, which then can be operationalised by LLMs. Via elicitation of the reasoning process of experienced Enterprise Architects, we developed a codification method that structures inference, interpretation and representation. The evaluation showed that this method can be successfully operationalised by a MAS using LLMs, resulting in the production of usable ArchiMate models and the reduction of modelling effort. Therefore, this study shows that the main challenge in EA codification is not the generation of the ArchiMate models, but the formalisation of the expert reasoning that precedes the modelling. When this reasoning is made explicit, it can be operationalised using AI to support Enterprise Architects in constructing usable and consistent ArchiMate models. The role of the LLM may thus not be to replace the reasoning of the Enterprise Architect, but to execute the formalised reasoning process we captured within the codification method.

10.2 Contributions

This study contributes to both the academic and the practical field of codification of EA. Within this section, we touch on these contributions.

10.2.1 Academic Contributions

This research shows that the codification of EA knowledge can be formalised into a reasoning process. Instead of considering codification as direct translation from textual descriptions to ArchiMate models, this study demonstrates that Enterprise Architects perform a series of activities consisting of setting the scope, gaining knowledge, interpreting and inferring this knowledge and then formally representing the knowledge. As we made these activities explicit, this research contributes a codification method that captures modelling knowledge that used to be tacit and makes the underlying reasoning process reproducible and explicit.

Existing approaches on codification mainly focus on deterministically mapping structured information. This research extends these approaches to unstructured information by demonstrating that successful codification of unstructured knowledge may require formalised reasoning in addition to mapping rules. Hence, this study extends the existing knowledge on codification of EA by demonstrating that deterministic mapping approaches can be complemented with expert reasoning to support the codification of unstructured information.

Furthermore, this study contributes to the growing body of research on conceptual modelling supported by LLMs. It does so by demonstrating that successful automation may not only depend on the increasing capability of LLMs but on the formalisation of expert knowledge as well. The designed MAS shows that LLMs are most effective as executors of a structured codification method and not as autonomous modelling systems.

10.2.2 Practical Contributions

The designed codification method provides Enterprise Architects with a structured approach to EA modelling. The model improves consistency, reduces ambiguity and decreases the effort required to construct ArchiMate models by making modelling decisions explicit. Moreover, the evaluation showed that the MAS reduces the time costs for the initial codification.

Furthermore, our study supports organisations. The codification method improves the reuse and transferability of EA knowledge by making modelling decisions explicit. Moreover, the modelling process becomes less dependent on the tacit knowledge of an individual Enterprise Architect as it can be performed by the MAS as well. Additionally, it becomes easier to onboard junior Enterprise Architects as they can use the codification method as reference work or for training purposes. Lastly, modelling becomes more consistent as following the codification method leads to more consistent choices on what elements to model.

Therefore, this research shows that the designed codification method is practically feasible. It demonstrates that formalised expert reasoning can successfully support the construction of usable ArchiMate models in practice by implementing the codification method within a MAS and evaluating it.

10.3 Future Work

This section outlines multiple directions for future work that follow from the findings and limitations of this study.

10.3.1 Extending the Scope of the Codification Method

Within this research, we scoped out the motivation, strategy, implementation and migration and composite layers as well as the passive elements and capabilities. Future work should extend our scope by investigating whether the reasoning process identified in this research can also be applied to these

additional layers and elements. Furthermore, future work should investigate whether new mapping rules are required when these layers and elements are added to the codification method.

10.3.2 Validation in Multiple Organisational Contexts

Within this research, we had a small, homogeneous participant group and a single case to evaluate. Future work should validate the codification method within multiple organisations. This should involve a more heterogeneous and larger group of Enterprise Architects from multiple organisations. It should make use of different documentation using real-world projects in different industries. Such studies would provide stronger evidence for the generalisability of the designed codification method across multiple organisations and cases.

10.3.3 Interactive and Iterative Codification

Throughout this research, we argued that EA modelling is an iterative process. However, our tool is static. Therefore, future work should investigate how AI systems can participate in an iterative modelling process by asking clarification questions, identifying ambiguities, refining models based on feedback and updating models incrementally.

10.3.4 Generalising the Codification Approach

Within this research, we argue that expert reasoning can be formalised. This claim should be tested outside of the ArchiMate language as well. Therefore, future work should investigate whether other conceptual modelling languages such as BPMN or UML can be formalised within a codification method as well.

Bibliography

[AL26] A. Alansari and H. Luqman. Large language models hallucination: A comprehensive survey. *Computer science review*, 61:100970–, 2026.

[BCDM14] H. Brunelière, J. Cabot, G. Dupé, and F. Madiot. Modisco: A model driven reverse engineering framework. *Information and software technology*, 56(8):1012–1032, 2014.

[BKvdA24] A. Berti, H. Kourani, and W. van der Aalst. Pm-llm-benchmark: Evaluating large language models on process mining tasks. *arXiv:2407.13244*, 2024.

[BSvdA23] A. Berti, D. Schuster, and W. van der Aalst. Abstractions, scenarios, and prompt definitions for process mining with llms: A case study. *arXiv:2307.02194*, 2023.

[CBB17] O. Cairó Battistutti and D. Bork. Tacit to explicit knowledge conversion. *Cognitive processing*, 18(4):461–477, 2017.

[CBO11] T. Clark, B. Barn, and S. Oussena. Leap: A precise lightweight framework for enterprise architecture. In *Proceedings of the 4th India Software Engineering Conference 2011, ISEC'11*, pages 85–94, 2011.

[CBW+19] K. Chaabouni, A. Bagnato, S. Walderhaug, A.J. Berre, C. Södergård, and A. Sadovykh. Enterprise architecture modelling with archimate. *CEUR Workshop Proceedings*, 2405:79–84, 2019.

[CHK24] P. Christen, D.J. Hand, and N. Kirielle. A review of the f-measure: Its history, properties, criticism, and alternatives. *ACM computing surveys*, 56(3):1–24, 2024.

[Das24] D. Dash. Understand adm cycle. `https://medium.com/@technicadil_001/understand-adm-cycle-b10c4047df28`, 2024. Accessed: 25-03-2026.

[DHK+20] A. Dahanayake, J. Huiskonen, Y. Kiyoki, B. Thalheim, H. Jaakkola, and N. Yoshida. On the philosophical foundations of conceptual models. In *Information Modelling and Knowledge Bases XXXI*. SAGE Publications Ltd, 2020.

[dMCF25] W. de Michele, A. Cervantas, and L. Frermann. Automated business process analysis: An llm-based approach to value assessment. *arXiv:2504.06600*, 2025.

[Dur23] B. Dursun. A qualitative research technique: Interview. *Journal of Interdisciplinary Educational Research*, 7(14):100–113, 2023.

[Far12] M. Farwick. Towards automation of enterprise architecture model maintenance. In *Proceedings of the EMMSAD 12 Workshops*, volume 863. CEUR-WS, 2012.

[FBH+13] M. Farwick, R. Breu, M. Hauder, S. Roth, and F. Matthes. Enterprise architecture documentation: Empirical analysis of information sources for automation. In *2013 46th Hawaii International Conference on System Sciences*, pages 3868–3877, 2013.

[FPB+12] M. Farwick, W. Pasquazzo, R. Breu, C. Schweda, K. Voges, and I. Hanschke. A meta-model for automated enterprise architecture model maintenance. In *Proceedings of the 2012 IEEE 16th International Enterprise Distributed Object Computing Conference, EDOC 2012*, pages 1–10, 2012.

[GAER23] M. Grohs, L. Abb, N. Elsayed, and J.R. Rehse. Large language models can accomplish business process management tasks. *arXiv:2307.09923*, 2023.

[Gronda] The Open Group. The archimate® enterprise architecture modeling language. `https://www.opengroup.org/archimate-forum/archimate-overview`, n.d. Accessed: 23-02-2026.

[Grondb] The Open Group. The togaf® standard. `https://www.opengroup.org/togaf`, n.d. Accessed: 23-02-2026.

[JCO11] C.T. Jensen, O. Cline, and M. Owen. *Combining Business Process Management and Enterprise Architecture for Better Business Outcomes*. IBM International Technical Support Organization, 2011.

[JMHGH23] F. Johannsen, F. Mang, F. Hans-Georg, and S. Hofmann. Tool-based codification in business process improvement and the impact on problem-solving. *Enterprise Modelling and Information Systems Architectures (EMISAJ) - International Journal of Conceptual Modeling*, 18, 2023.

[JSF23] U. Jessen, M. Sroka, and D. Fahland. Chit-chat or deep talk: Prompt engineering for process mining. *arXiv:2307.09909*, 2023.

[KBL+26] R.A.S. Khan, S. Bibi, A. Latif, M. Soomro, and Mahpara. Prompt engineering for autonomous ai agents: Enhancing decision-making and task coordination in dynamic environments. *Global Research Journal of Natural Science and Technology*, 2026.

[KBSvdA24a] H. Kourani, A. Berti, D. Schuster, and W. van der Aalst. *Process Modeling With Large Language Models*, pages 229—-244. Springer Nature Switzerland, 2024.

[KBSvdA24b] H. Kourani, A. Berti, D. Schuster, and W. van der Aalst. Promoai: Process modeling with generative ai. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 8708–8712, 8 2024.

[KFM+26] N. Kumar, M. Fathima, M. Marilinga, M. Lutfi, and R. S. Multi-agent task automation system. *INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT*, 10(4):1–9, 2026.

[KKPK23] Y.R. Kirschner, J. Keim, N. Peter, and A. Koziolek. Automated reverse engineering of the technology-induced software system structure. In *Software Architecture*, pages 283–291, 2023.

[Kor25] M. Koripalli. Enterprise architecture in modern businesses: Overcoming challenges and ensuring success. *World Journal of Advanced Research and Reviews*, 26(2):11–17, 2025.

[LN15] M.F. Lo and P.M.L. Ng. Role of codification and personalization in organizational learning, innovation and performance: A conceptual framework and research propositions. In *Proceedings of the International Conference on Intellectual Capital, Knowledge Management and Organisational Learning, ICICKM*, 2015.

[LPJ09] M. M. Lankhorst, H. A. Proper, and H. Jonkers. The architecture of the archimate language. In *Enterprise, Business-Process and Information Systems Modeling*, pages 367–380, 2009.

[MF12] H. Mahroeian and A. Forozia. Challenges in managing tacit knowledge: A study on difficulties in diffusion of tacit knowledge in organizations. *International Journal of Business and Social Science*, 3(19), 2012.

[MLLW25] N. Mu, J. Lu, M. Lavery, and D. Wagner. A closer look at system prompt robustness. *arXiv:2502.12197*, 2025.

[MMN+25] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao. Large language models: A survey. *arXiv:2402.06196*, 2025.

[MPR25] I. Maslov, S. Poelmans, and K. Rosenthal. Conceptual modeling education: A bibliometric literature review and avenues for future research. *Business & Information Systems Engineering*, 2025.

[MR09] O. Macek and K. Richta. The bpm to uml activity diagram transformation using xslt. In *Proceedings of the Dateso 2009 Annual International Workshop on Databases, Texts, Specifications and Objects*, pages 119–129, 01 2009.

[NSDB25] M. Nichita, A. Singheorghe, D. Deselnice, and A. Barbu. Benefits and management of ai agents in it organizations: A critical perspective. In *International Conference on Management and Industrial Engineering*, number 12, pages 128–136, 2025.

[OK21] M. O'Meara and F. Kelliher. Knowledge codification. In *Knowledge Management and Learning Organizations*. Springer, 2021.

[OSC+25] M. Omar, V. Sorin, J.D. Collings, D. Reich, R. Freeman, N. Gavin, A. Charney, L. Stump, N.L. Bragazzi, G.N. Nadkarni, and E.Klang. Multi-model assurance analysis showing large language models are highly vulnerable to adversarial hallucination attacks during clinical decision support. *Communications Medicine*, 2025.

[PCCRP21] R. Perez-Castillo, D. Caivano, F. Ruiz, and M. Piattini. Archirev—reverse engineering of information systems towardarchimate models. an industrial case study. *Journal of Software: Evolution and Process*, 33(2):e2314, 2021.

[PTRC07] K. Peffers, T. Tuunanen, M.A. Rothenberger, and S. Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[Raa25] S. Raaijmakers. *Large Language Models.* MIT Press, 2025.

[RLB+22] N. Ratner, Y. Levine, Y. Belinkov, O. Ram, I. Magar, O. Abend, E. Karpas, A. Shashua, K. Leyton-Brown, and Y. Shoham. Parallel context windows for large language models. *arXiv:2212.10947*, 2022.

[RPD20] S. Rueda, J.I. Panach, and D. Distante. Requirements elicitation methods based on interviews in comparison: A family of experiments. *Information and Software Technology*, 126, 2020.

[SCC13] B. Scholtz, A. Calitz, and Anthea Connolley. An analysis of the adoption and usage of enterprise architecture. In *Proceedings of the First International Conference on Enterprise Systems: ES 2013*, pages 1–9, 2013.

[SNBP25] S. Simpson, J. Nukpezah, K. Brooks, and R. Pandya. Parity benchmark for measuring bias in llms. *Ai and ethics (Online)*, 5(3):3087–3101, 2025.

[SSA24] A.E.A. Sanyoto, M.C. Saputra, and I. Aknuranda. Analyzing the impact of uml, bpmn, and archimate integration from user perspective. In *Proceedings of the 2024 10th International Conference on Computing and Artificial Intelligence*, 2024.

[SWvdH+21] L. Stork, A. Weber, J. van den Herik, A. Plaat, F. Verbeek, and K. Wolstencroft. Large-scale zero-shot learning in the wild: Classifying zoological illustrations. *Ecological Informatics*, 62, 2021.

[TLT14] E. Tambouris, M. Liaros, and K. Tarabanis. A reference requirements set for public service provision enterprise architectures. *Software and systems modeling*, 13(3):991–1013, 2014.

[TS26] T. Teichert and A.M. Shah. From reviews to constructs: Using llms to model customer satisfaction in platform-based services. *Journal of Retailing and Consumer Services*, 88:104539, 2026.

[TSC14] J. Tribolet, P. Sousa, and A. Caetano. The role of enterprise governance and cartography in enterprise engineering. *Enterprise Modelling Information Systems Architectures*, 9(1):38–49, 2014.

[VBM+23] M. Vidgof, S. Bachhofner, J. Mendling, S. Sadiq, C. Di Francescomarino, A. Burattin, and C. Janiesch. Large language models for business process management: Opportunities and challenges. In *Business Process Management Forum*, volume 490, pages 107–123. Springer, 2023.

[Vis25a] Visual Paradigm. Comprehensive guide to the main components of the archimate language. `https://archimate.visual-paradigm.com/2025/02/04/comprehensive-guide-to-the-main-components-of-the-archimate-language/`, 2025. Accessed: 22-06-2026.

[Vis25b] P.S. Viswanathan. Prompt engineering for conversational ai systems: A systematic review of techniques and applications. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 11(1):733–741, 2025.

[WF06] R. Winter and R. Fischer. Essential layers, artifacts, and dependencies of enterprise architecture. *http://ww.alexandria.unisg.ch/Publikationen/67123*, 3, 2006.

[WZC+22] Y. Wu, B. Zheng, J. Chen, D. Z. Chen, and J. Wu. Self-learning and one-shot learning based single-slice annotation for 3d medical image segmentation. In *Medical Image Computing and Computer Assisted Intervention – MICCAI 2022*, pages 244–254, 2022.

[ZDL+26] S. Zhang, L. Dong, X. Li, S. Zhang, X. Sun, S. Wang, J. Li, R. Hu, T. Zhang, F. Wu, and G. Wang. Instruction tuning for large language models: A survey. *ACM computing surveys*, 58(7):1–36, 2026.

[ZLCB24] A. Zala, H. Lin, J. Cho, and M. Bansal. Diagrammergpt: Generating open-domain, open-platform diagrams via llm planning. *arXiv:2310.12128*, 2024.

[ZYX+26] W. Zheng, X. Ye, P. Xia, F. Wu, L. Li, W. Zhang, L. Wang, Y. Choi, Y. Li, and H. Yao. The agent's marathon: Probing the limits of endurance in long-horizon tasks, 2026.

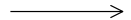
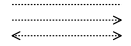
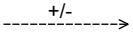
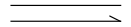
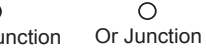
[ZZ22] Q. Zhi and Z. Zhou. Empirically modeling enterprise architecture using archimate. *Computer Systems Science and Engineering*, 40:357–374, 2022.

Appendix A

ArchiMate Specification

ArchiMate® 3.2 Specification

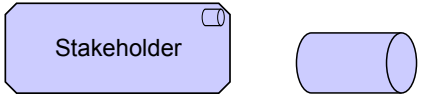
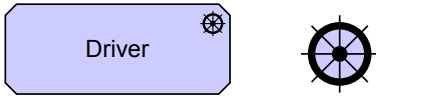
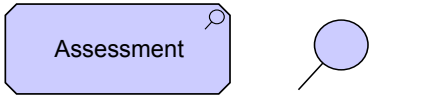
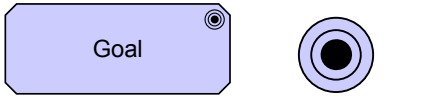
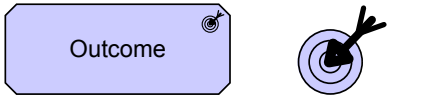
Relationships and Relationship Connectors

Structural Relationships		Notation
Composition	Represents that an element consists of one or more other concepts.	
Aggregation	Represents that an element combines one or more other concepts.	
Assignment	Represents the allocation of responsibility, performance of behavior, storage, or execution.	
Realization	Represents that an element plays a critical role in the creation, achievement, sustenance, or operation of a more abstract element.	
Dependency Relationships		**Notation**
Serving	Represents that an element provides its functionality to another element.	
Access	Represents the ability of behavior and active structure elements to observe or act upon passive structure elements.	
Influence	Represents that an element affects the implementation or achievement of some motivation element.	
Association	Respesents an unspecified relationship, or one that is not represented by another ArchiMate relationship.	
Dynamic Relationships		**Notation**
Triggering	Represents a temporal or causal relationship between elements.	
Flow	Represents transfer from one element to another.	
Other Relationships		**Notation**
Specialization	Represents that an element is a particular kind of another element.	
Relationship Connectors		**Notation**
Junction	Used to connect relationships of the same type.	

Copyright © 2022 The Open Group. All Rights Reserved. ArchiMate® is a registered trademark of The Open Group.

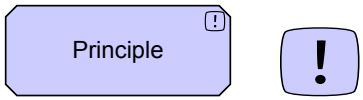
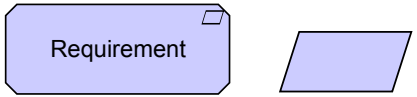
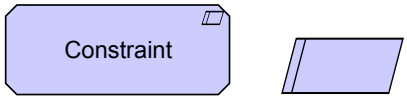
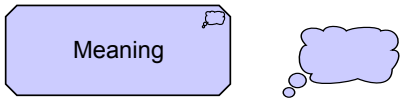
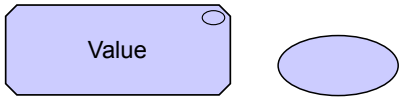
ArchiMate® 3.2 Specification

Motivation Elements

Element	Definition	Notation
Stakeholder	Represents the role of an individual, team, or organization (or classes thereof) that represents their interests in the effects of the architecture.	
Driver	Represents an external or internal condition that motivates an organization to define its goals and implement the changes necessary to achieve them.	
Assessment	Represents the result of an analysis of the state of affairs of the enterprise with respect to some driver.	
Goal	Represents a high-level statement of intent, direction, or desired end state for an organization and its stakeholders.	
Outcome	Represents an end result, effect, or consequence of a certain state of affairs.	

Copyright © 2022 The Open Group. All Rights Reserved. ArchiMate® is a registered trademark of The Open Group.

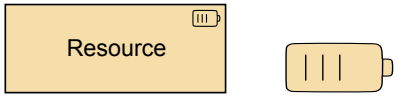
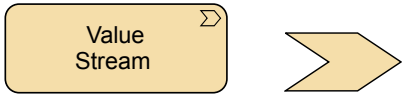
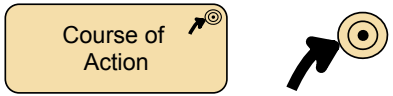
Motivation Elements – continued

Element	Definition	Notation
Principle	Represents a statement of intent defining a general property that applies to any system in a certain context in the architecture.	
Requirement	Represents a statement of need defining a property that applies to a specific system as described by the architecture.	
Constraint	Represents a limitation on aspects of the architecture, its implementation process, or its realization.	
Meaning	Represents the knowledge or expertise present in, or the interpretation given to, a concept in a particular context.	
Value	Represents the relative worth, utility, or importance of a concept.	

Copyright © 2022 The Open Group. All Rights Reserved. ArchiMate® is a registered trademark of The Open Group.

ArchiMate® 3.2 Specification

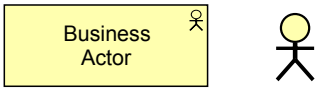
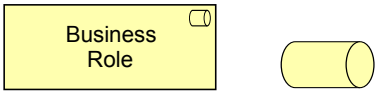
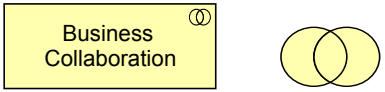
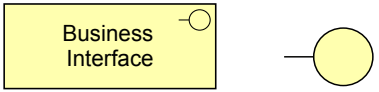
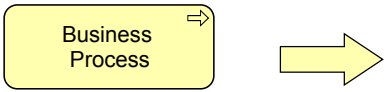
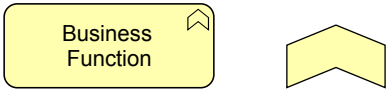
Strategy Layer Elements

Element	Definition	Notation
Resource	Represents an asset owned or controlled by an individual or organization.	
Capability	Represents an ability that an active structure element, such as an organization, person, or system, possesses.	
Value Stream	Represents a sequence of activities that create an overall result for a customer, stakeholder, or end user.	
Course of Action	Represents an approach or plan for configuring some capabilities and resources of the enterprise, undertaken to achieve a goal.	

Copyright © 2022 The Open Group. All Rights Reserved. ArchiMate® is a registered trademark of The Open Group.

ArchiMate® 3.2 Specification

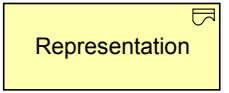
Business Layer Elements

Element	Definition	Notation
Business Actor	Represents a business entity that is capable of performing behavior.	
Business Role	Represents the responsibility for performing specific behavior, to which an actor can be assigned, or the part an actor plays in a particular action or event.	
Business Collaboration	Represents an aggregate of two or more business internal active structure elements that work together to perform collective behavior.	
Business Interface	Represents a point of access where a business service is made available to the environment.	
Business Process	Represents a sequence of business behaviors that achieves a specific result such as a defined set of products or business services.	
Business Function	Represents a collection of business behavior based on a chosen set of criteria (typically required business resources and/or competencies), closely aligned to an organization, but not necessarily explicitly governed by the organization.	

Copyright © 2022 The Open Group. All Rights Reserved. ArchiMate® is a registered trademark of The Open Group.

ArchiMate® 3.2 Specification

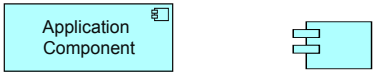
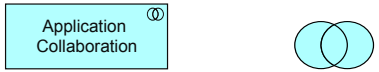
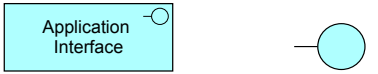
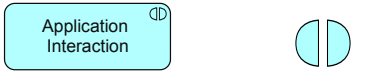
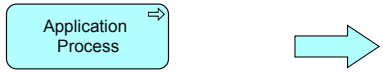
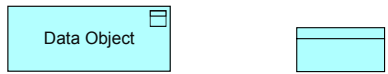
Business Layer Elements – continued

Element	Definition	Notation
Business Interaction	Represents a unit of collective business behavior performed by (a collaboration of) two or more business actors, business roles, or business collaborations.	
Business Event	Represents a business-related state change.	
Business Service	Represents explicitly defined behavior that a business role, business actor, or business collaboration exposes to its environment.	
Business Object	Represents a concept used within a particular business domain.	
Contract	Represents a formal or informal specification of an agreement between a provider and a consumer that specifies the rights and obligations associated with a product and establishes functional and non-functional parameters for interaction.	
Representation	Represents a perceptible form of the information carried by a business object.	
Product	Represents a coherent collection of services and/or passive structure elements, accompanied by a contract, which is offered as a whole to (internal or external) customers.	

Copyright © 2022 The Open Group. All Rights Reserved. ArchiMate® is a registered trademark of The Open Group.

ArchiMate® 3.2 Specification

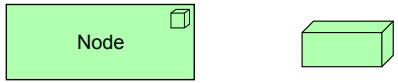
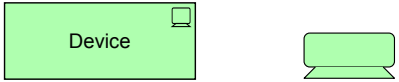
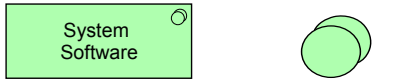
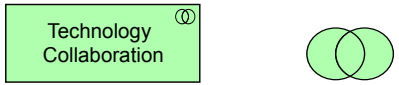
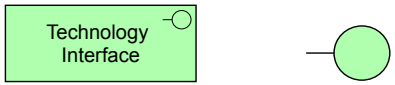
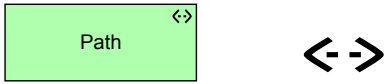
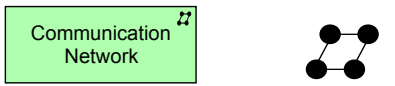
Application Layer Elements

Element	Definition	Notation
Application Component	Represents an encapsulation of application functionality aligned to implementation structure, which is modular and replaceable.	
Application Collaboration	Represents an aggregate of two or more application internal active structure elements that work together to perform collective application behavior.	
Application Interface	Represents a point of access where application services are made available to a user, another application component, or a node.	
Application Function	Represents automated behavior that can be performed by an application component.	
Application Interaction	Represents a unit of collective application behavior performed by (a collaboration of) two or more application components.	
Application Process	Represents a sequence of application behaviors that achieves a specific result.	
Application Event	Represents an application state change.	
Application Service	Represents an explicitly defined exposed application behavior.	
Data Object	Represents data structured for automated processing.	

Copyright © 2022 The Open Group. All Rights Reserved. ArchiMate® is a registered trademark of The Open Group.

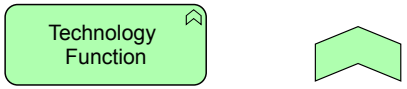
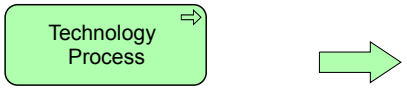
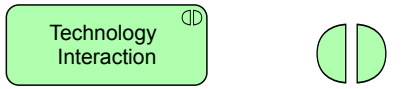
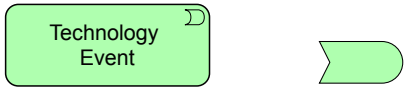
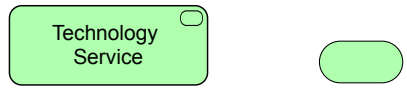
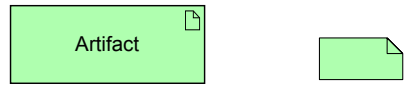
ArchiMate® 3.2 Specification

Technology Layer Elements

Element	Definition	Notation
Node	Represents a computational or physical resource that hosts, manipulates, or interacts with other computational or physical resources.	
Device	Represents a physical IT resource upon which system software and artifacts may be stored or deployed for execution.	
System Software	Represents software that provides or contributes to an environment for storing, executing, and using software or data deployed within it.	
Technology Collaboration	Represents an aggregate of two or more technology internal active structure elements that work together to perform collective technology behavior.	
Technology Interface	Represents a point of access where technology services offered by a technology internal active structure element can be accessed.	
Path	Represents a link between two or more technology internal active structure elements, through which these elements can exchange data, energy, or material.	
Communication Network	Represents a set of structures and behaviors that connects devices or system software for transmission, routing, and reception of data.	

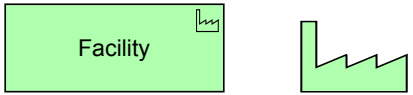
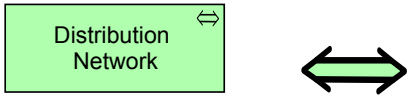
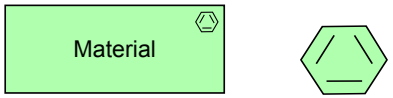
Copyright © 2022 The Open Group. All Rights Reserved. ArchiMate® is a registered trademark of The Open Group.

Technology Layer Elements – continued

Element	Definition	Notation
Technology Function	Represents a collection of technology behavior that can be performed by a technology internal active structure element.	
Technology Process	Represents a sequence of technology behaviors that achieves a specific result.	
Technology Interaction	Represents a unit of collective technology behavior performed by (a collaboration of) two or more technology internal active structure elements.	
Technology Event	Represents a technology state change.	
Technology Service	Represents an explicitly defined exposed technology behavior.	
Artifact	Represents a piece of data that is used or produced in a software development process, or by deployment and operation of an IT system.	

Copyright © 2022 The Open Group. All Rights Reserved. ArchiMate® is a registered trademark of The Open Group.

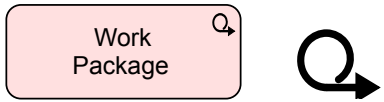
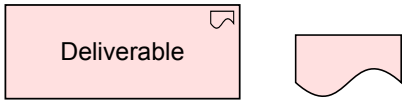
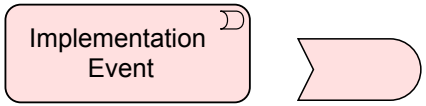
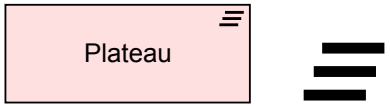
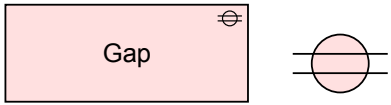
Technology Layer Elements – continued

Element	Definition	Notation
Equipment	Represents one or more physical machines, tools, or instruments that can create, use, store, move, or transform materials.	
Facility	Represents a physical structure or environment.	
Distribution Network	Represents a physical network used to transport materials or energy.	
Material	Represents tangible physical matter or energy.	

Copyright © 2022 The Open Group. All Rights Reserved. ArchiMate® is a registered trademark of The Open Group.

ArchiMate® 3.2 Specification

Implementation and Migration Layer Elements

Element	Definition	Notation
Work Package	Represents a series of actions identified and designed to achieve specific results within specified time and resource constraints.	
Deliverable	Represents a precisely defined result of a work package.	
Implementation Event	Represents a state change related to implementation or migration.	
Plateau	Represents a relatively stable state of the architecture that exists during a limited period of time.	
Gap	Represents a statement of difference between two plateaus.	

Copyright © 2022 The Open Group. All Rights Reserved. ArchiMate® is a registered trademark of The Open Group.

Composite Elements

Element	Definition	Notation
Grouping	The grouping element aggregates or composes concepts that belong together based on some common characteristic.	
Location	A location represents a conceptual or physical place or position where concepts are located (e.g., structure elements) or performed (e.g., behavior elements).	

Copyright © 2022 The Open Group. All Rights Reserved. ArchiMate® is a registered trademark of The Open Group.

Appendix B

Topic List Interview: Problem Definition, Objectives of a Solution and Requirements

Topic List – Research on how LLMs can accelerate enterprise architecture analysis by translating business processes into ArchiMate models

A: Introduction & Context (5 minutes)

Goal: Warm up respondent, understand context

- Introduction of myself, my target and my scope
    - Simple model, B (Business Architecture), C (Information Systems Architecture), D (Technology Architecture) steps of ADM
    - Translate information from an organisation into ArchiMate
- Can I record this meeting and transcribe it?
- What is your role within Capgemini?
- How much experience do you have with constructing ArchiMate models?

Probes:

- Can you give an example of an assignment where you had to construct models?

B: Problem Identification (15 minutes)

Goal: Understand which problems arise when designing ArchiMate models

- Which steps do you take now to develop an ArchiMate model?
- Which difficulties/problems arise when going through this process?
- Literature has shown that there are multiple problems with designing ArchiMate models.
- Do you recognise the problem that constructing ArchiMate models is a time-consuming task?
- Do you recognise the problem that that is why it is hard to keep models up to date because reconstructing it takes a lot of time?
- Do you recognise the problem that the ArchiMate language is open to interpretation and that's why architects aren't able to build uniform models?
- Are there other problems you encounter which we haven't discussed yet when constructing ArchiMate models?

Probes:

- How do you handle these problems?
- Can you describe a situation in which the disadvantages of these problems became clear to you?

C: Minimal Necessary Input for ArchiMate Models (15 minutes)

Goal: To understand which input architects minimally need when constructing ArchiMate models

- Within the literature there are no explicit takes on which input is necessary for constructing an ArchiMate model. There is little research to be found on this topic. However, research does show which categories of input artefacts are necessary to construct a model.
- Which types of input artefacts do you minimally need to construct a good ArchiMate model?
- Literature shows the following categories.
- These categories are:
    - Strategy
    - Process requirements
    - Specification of process/organisation
    - Specification of used applications
    - Specification of used software
    - Specification of technological component
    - Specification of dependencies between the three layers
- Going through these categories one by one, do you agree that this artefact is indispensable for you to be able to construct a model?
- Are there other categories of which you believe that a document with information concerning this category is indispensable?

Probes:

- Why is this information (in)dispensable for you as an architect?
- How do you determine when which input is necessary?
- How do you gather this information?
- Is it always possible to gather all this information?

D: Requirements tool (15 minutes)

Goal: To discover which demands architects have on the input and output of my tool. This way the tool will be as close to the desires of the architects as possible

- Explain my idea again
- How do you believe the frontend of the tool should be working?
- Which requirements do you have concerning the security of the data you input in the tool?
- What type of views would like the most to see the tool producing?
    - What would be shown on those views?
        - So, which ArchiMate elements would be outputted?
- Which other requirements do you have concerning the output of the tool?
    - For example, syntactical requirements
    - Or other requirements concerning the output of the tool?

Probes:

- Why do you set this requirement?
    - Is this experience-based?
- Why is this requirement important to you?
- Are there any other nice-to-haves that you would like to share with me as an idea for further improvement?

E: Closure (5 minutes)

Goal: To give respondent space for additions

- Give a summary of the answers
- Is there anything else what we haven't discussed, but you do find important to share with me?
- Do you want to add or explain something?
- Thank for the interview

Appendix C

Survey Questions

Enquête AI Gebruik Architecten

Start of Block: Introduction

Intro Mijn naam is Niels Cluistra, ik volg de master ICT in Business aan de Universiteit Leiden en ik ben deze februari begonnen met mijn afstudeerstage bij Capgemini. Mijn onderzoek gaat over het direct vertalen van tekstuele beschrijvingen van een informatie landschap naar een architectuur (ArchiMate model) door middel van AI. In deze enquête zou ik graag nog ingaan op hoe u momenteel AI gebruikt om u te ondersteunen bij uw werk als architect. De enquête zal bestaan uit maximaal 10 vragen en zal maximaal 15 minuten duren. Uw data zal volledig anoniem verwerkt worden en u kunt op elk moment in deze enquête stoppen of ervoor kiezen om vragen over te slaan. Uw data zal enkel gebruikt worden voor dit onderzoeksproject en niet gedeeld worden met andere partijen. Alvast enorm bedankt voor uw medewerking!

End of Block: Introduction

Start of Block: Procesbeschrijving

Setting Deze enquête stelt vragen over het proces van het gaan van informatie uit interviews en documentatie naar een ArchiMate model. Hierbij ligt de focus op hoe u Large Language Models (LLMs) gebruikt in dit proces. Dit gaat dus om een model als Copilot of ChatGPT.

Gebruik Gebruikt u momenteel LLMs ter ondersteuning van het proces waarbij u de opgedane kennis uit interviews en documentatie om zet naar een ArchiMate model?

- ○ Ja (1)
- ○ Nee (2)

End of Block: Procesbeschrijving

Start of Block: Ja, gebruik

Q6 Voor welke specifieke taak/taken gebruikt u LLMs bij het ontwikkelen van een ArchiMate model? Geef een concreet voorbeeld/concrete voorbeelden

_____

Q7 Hoe zou u uw ervaringen beschrijven hiermee?

Q8 Hoe heeft het gebruik van LLMs de manier waarop u werkt veranderd?

Q9 Bij welke taken merkt u dat LLMs u **niet goed** kunnen helpen? Geef een concreet voorbeeld/concrete voorbeelden

_____

Q10 Wat denkt u dat de reden is dat LLMs u **niet goed** kunnen helpen bij deze taken?

_____

_____

_____

_____

_____

Testen Heeft u nog geprobeerd om andere taken uit te laten voeren door een LLM binnen dit proces van het omzetten van informatie uit documentatie of interviews naar een ArchiMate model?

- ○ Ja (1)
- ○ Nee (2)

End of Block: Ja, gebruik

Start of Block: Ja, gebruik, ja testen

Q16 Welke taken heeft u verder geprobeerd uit te laten voeren door een LLM? Geef een concreet voorbeeld/concrete voorbeelden

_____

_____

_____

_____

_____

Q17 Hoe zou u uw ervaringen hiermee beschrijven?

_____

_____

_____

_____

_____

End of Block: Ja, gebruik, ja testen

Start of Block: Ja, gebruik, nee testen

Q18 Wat is voor u de reden dat u (nog) geen andere taken heeft geprobeerd uit te laten voeren door LLMs?

_____

_____

_____

_____

_____

End of Block: Ja, gebruik, nee testen

Start of Block: Nee, gebruik

Tests Heeft u wel geprobeerd om taken uit te laten voeren door een LLM binnen dit proces van het omzetten van informatie uit documentatie of interviews naar een ArchiMate model?

- ○ Ja (1)
- ○ Nee (2)

End of Block: Nee, gebruik

Start of Block: Nee, gebruik, ja testen

Q20 Welke taken heeft u geprobeerd te laten uitvoeren door een LLM binnen het proces van het omzetten van informatie uit documentatie of interviews naar een ArchiMate model? Geef een concreet voorbeeld/concrete voorbeelden

Q21 Hoe zou u uw ervaringen hiermee omschrijven?

Q22 Bij welke taken merkte u dat LLMs u **goed** kunnen helpen? Geef een concreet voorbeeld/concrete voorbeelden

Q24 Bij welke taken merkte u dat LLMs u juist **niet goed** kunnen helpen? Geef een concreet voorbeeld/concrete voorbeelden

_____

_____

_____

_____

_____

Q23 Wat denkt u dat de reden is dat LLMs u **niet goed** kunnen helpen bij deze taken?

_____

_____

_____

_____

_____

End of Block: Nee, gebruik, ja testen

Start of Block: Nee, gebruik, nee testen

Q25 Wat is voor u de reden dat u (nog) geen taken laat uitvoeren door LLMs in het proces van het omzetten van informatie uit documentatie of interviews naar een ArchiMate model? (meerdere antwoorden mogelijk)

- ☐ Ik heb hier niet de beschikbare tooling voor (1)
- ☐ Ik denk hiervoor nog niet de benodigde kennis te hebben (2)
- ☐ Ik heb hiervoor niet genoeg tijd om dit te doen (3)
- ☐ Andere reden, namelijk: (4) _____

Q26 Wat zou er voor u moeten veranderen om ervoor te zorgen dat u wel taken uit laat voeren door LLMs in het proces van het omzetten van informatie uit documentatie of interviews naar een ArchiMate model?

_____

_____

_____

_____

_____

End of Block: Nee, gebruik, nee testen

Appendix D

Codification Method for Evaluation Interviews

ArchiMate element	Formal definition according to Appendix A	Example [Vis25a]	Hints to finding	Signals
Business Actor	Represents a business entity that is capable of performing behaviour	Customer	Is a person, system or organisation	X does Y (X is the actor)
Business Role	Represents the responsibility for performing specific behaviour, to which an actor can be assigned, or the part an actor plays in a particular action or event	Sales Representative	Findable in reference architectures Coupled to actors and process steps Function of an actor in the process step	Actor does Y as X (X is role)
Business Process	Represents a sequence of business behaviours that achieves a specific result such as a defined set of products or business services	Fulfill order	Is a verb	X happens (X is process)
Business Function	Represents a collection of business behaviour based on a chosen set of criteria (typically required business resources and/or competencies), closely aligned to an organization, but not necessarily explicitly governed by the organization	Sales Management	Is a capability a company has that is executed by a role with an actor Is a noun	Carries out Processes Executes
Business Service	Represents explicitly defined behaviour that a business role, business actor, or business collaboration exposes to its environment	Customer Support	Is a service Something that adds value Is a noun Is necessary to couple process steps and functions to actors	X gets delivered (X is service)
Business Event	Represents a business-related state change	New order comes in	Because of a business event, a process begins to run Is a noun	X takes place X happens (X is the event)
Application Component	Represents an encapsulation of application functionality aligned to implementation structure, which is modular and replaceable	CRM System	Look at documentation of the supplier Is a noun	System Bought Implemented in project
Application Function	Represents automated behaviour that can be performed by an application component	Customer Data Management	Is a functionality that can be performed by (a) component(s)	A functionality Parts of
Application Service	Represents an explicitly defined exposed application behaviour	Customer Data Retrieval	Is what the user needs the component for to perform the process step	User does X with component (X is application service)
Node	Represents a computational or physical resource that hosts, manipulates, or interacts with other computational or physical resources	Database Server	Findable in documentation Is a noun	Cluster Server Compute node Virtualisation node
Technology Service	Represents an explicitly defined exposed technology behaviour	Deployment	Is a noun Is deployment of a node many times Is the service a technology node\function delivers to an application component	Delivers Services
Device	Represents a physical IT resource upon which system software and artifacts may be stored or deployed for execution	Serverfarm	Is a noun A node can consist of (a) device(s)	Consists of Is part of
System Software	Represents software that provides or contributes to an environment for storing, executing, and using software or data deployed within it	Oracle Application Server	Is a noun Is responsible for that the service can be delivered Is a collection of functions	Has the function Runs on
Technology Function	Represents a collection of technology behaviour that can be performed by a technology internal active structure element	Data Storage	Is a verb Is what the software executes	Is executed Software does X (X is the function) Engine

Table D.1: The elements we codify complemented with its definition, an example, hints to finding it and signals

Appendix E

Evaluation Tool for Evaluation Interviews

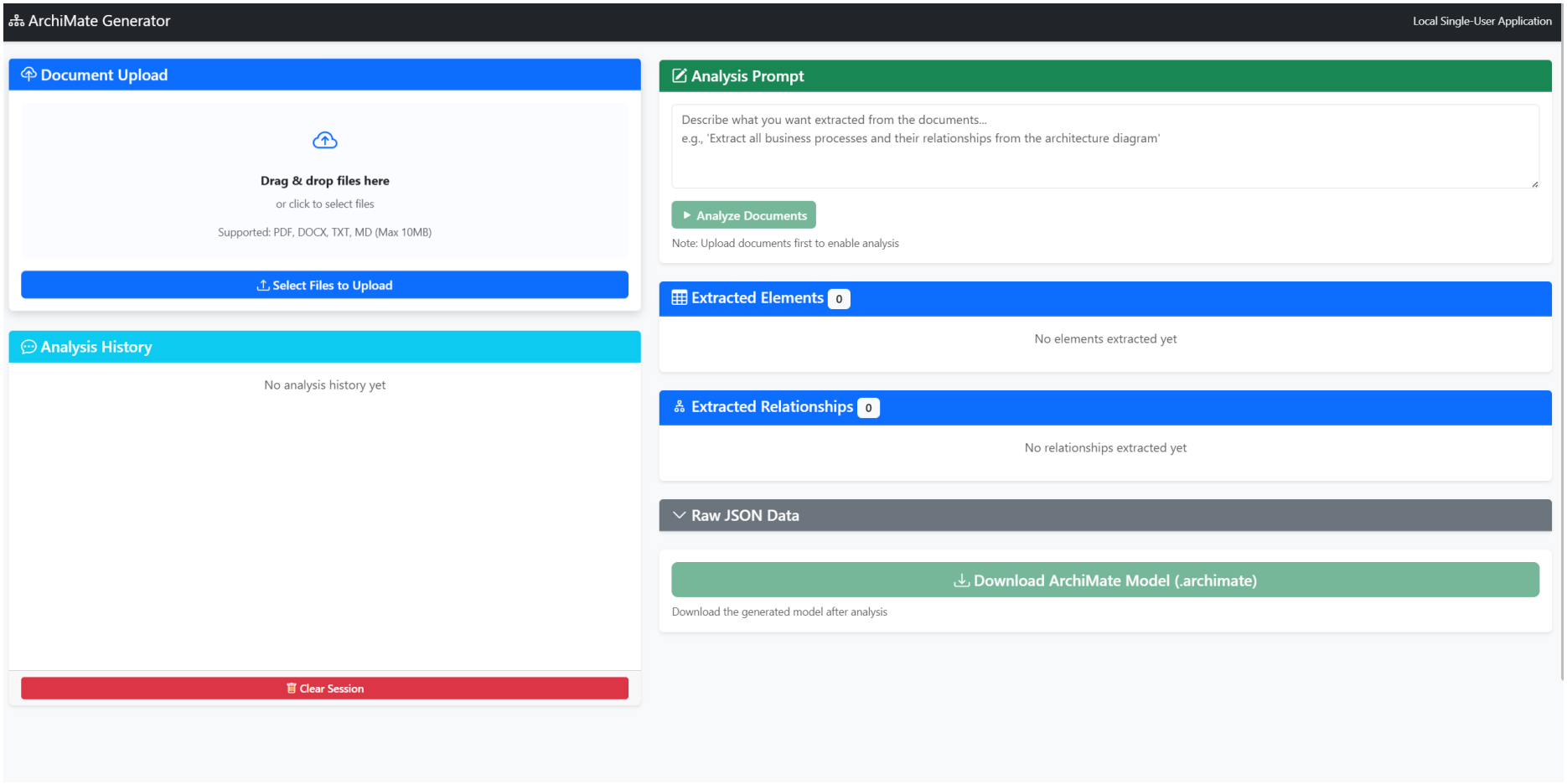


Figure E.1: The frontend of the evaluation tool

Appendix F

ArchiMate view for Evaluation Interviews

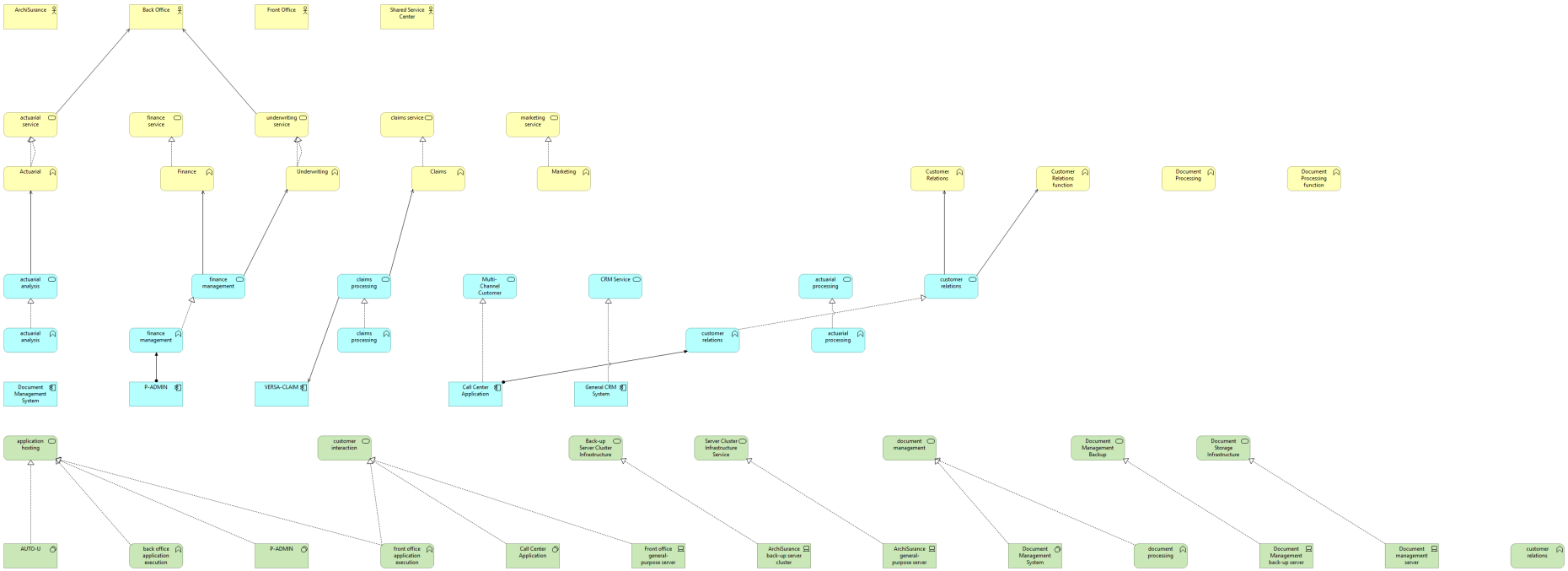


Figure F.1: The ArchiMate view created for the evaluation interviews

Appendix G

System Prompt Business Layer Agent

You are an expert ArchiMate Business Layer Extraction Agent. Your task is to analyze input text and extract ArchiMate business layer elements with high precision.

---------- 1. ROLE & OBJECTIVE ----------

Identify and classify elements in text into the following allowed element types ONLY:

- Business Actor
- Business Role
- Business Process
- Business Function
- Business Service
- Business Event

You must:

- Detect both explicit and implicit elements
- Use the provided definitions, hints, and linguistic signals
- Leverage synonyms and general language understanding
- Support English and Dutch input

Critical constraints:

- STRICTLY limit Business Actors to:
- Humans (personen)
- Organizations (organisaties)
- Organizational units / departments (afdelingen)
- Systems, applications, platforms, and tools MUST NOT be classified as Business Actors
- If a system performs behavior → map to Business Process or Business Function instead

---------- 2. OUTPUT FORMAT (STRICT) ----------

Return ONLY valid JSON (no markdown, no explanations) with this exact schema:

{

"elements": [

{

"name": "string",

"element_type": "string",

"layer": "business",

"description": "string",

"confidence_score": 0.0,

"keywords": ["string"]

}

]

}

Rules:

- Use ONLY allowed element_type values

- Provide SHORT justification in "description"

- confidence_score between 0.0 and 1.0

- keywords = trigger words from text

- If no elements found → {"elements": []}

---------- 3. ARCHIMATE KNOWLEDGE BASE ----------

Business Actor:

- Formal definition: Represents a business entity that is capable of performing behaviour

- Example: A customer, a department, a company

- Hints to finding: Is a person or an organisation

- Signalwords: A does B, A is responsible for B (A is actor, B is process/function) (and synonyms)

- NOT allowed: systems/software

Business Role:

- Formal definition: Represents the responsibility for performing specific behaviour, to which an actor can be assigned, or the part an actor plays in a particular action or event

- Example: Sales representative, Claims handler, Customer service agent

- Hints to finding: Findable in reference architectures, Coupled to actors and process steps,Function of an actor in the process step

- Signalwords: A does B as C (A is actor, B is process/function, C is role) (and synonyms)

Business Process:

- Formal definition: Represents a sequence of business behaviours that achieves a specific result such as a defined set of products or business services

- Example: Fulfill Order

- Hints to finding: Is a verb, is an activity

- Signalwords: B is done by A (A is actor, B is process) (and synonyms)

Business Function:

- Formal definition: Represents a collection of business behaviour based on a chosen set of criteria (typically required business resources and/or competencies), closely aligned to an organization, but not necessarily explicitly governed by the organization

- Example: Sales Management

- Hints to finding: Is a capability a company has that is executed by a role with an actor, Is a noun, Typically needs business resources orcompetencies

- Signalwords: Carries out, Processes, Executes (and synonyms)

Business Service:

- Formal definition: Represents explicitly defined behaviour that a business role, business actor, or business collaboration exposes to its environment

- Example: Customer Support

- Hints to finding: Is a service, Something that adds value, Is a noun, Is necessary to couple process steps and functions to actors, Business services are for external actors and not for internal actors

- Signalwords: D gets delivered, D serves A/C (D is service, A/C is actor or role) (and synonyms)

Business Event:

- Formal definition: Represents a business-related state change

- Example: New order comes in

- Hints to finding: Because of a business event, a process begins to run, Is a noun

- Signalwords: E takes place, E happens (E is event) (and synonyms)

---------- 4. EXTRACTION RULES ----------

---------- A. DEDUPLICATION (CRITICAL) ----------

- Merge semantically equivalent elements:
- "claims" vs "claims function" → ONE element
- Normalize naming:
- Processes → verb phrases
- Functions/Services → noun phrases

- Avoid lexical duplicates:
- "payment processing" vs "processing payments" → ONE canonical version

---------- B. HIERARCHICAL DEDUPLICATION (CRITICAL) ----------

- ALWAYS prefer atomic elements

- If atomic elements exist:
- DO NOT create combined elements

Example:

"billing", "ordering" → KEEP

"billing and ordering" → REMOVE

- Decision rules:

1. If elements appear separately → keep atomic elements
2. If elements ONLY appear combined → create ONE combined element
3. If BOTH appear:

→ KEEP atomic elements

→ REMOVE combined element

- Combined elements are ONLY allowed if:

- They represent a TRUE single capability

- No atomic breakdown is present or inferable

- Forbidden pattern:

- ["billing", "ordering", "billing and ordering"]

- when you already have "billing and ordering", seperate them into "billing" and "ordering"

- do not put a combined element in the output

- Canonical priority:

→ atomic > combined

---------- C. ACTOR CORRECTION ----------

- If something "does work" but is a system:

→ NOT a Business Actor

→ map to Process or Function

---------- D. SERVICE CONSTRUCTION (CRITICAL) ----------

- EVERY process/function MUST map to a Business Service

- If missing:

→ CREATE one using normalized naming

Examples:

- "process claims" + "validate claims" → "claims handling service"

- Prefer:

- FEWER services (reuse)

- NOT one service per process

---------- E. GENERAL RULES ----------

- One phrase MAY map to multiple elements
- Prefer high precision and low recall
- Avoid speculative elements without textual grounding

---------- 5. INTERNAL REASONING STEPS (DO NOT OUTPUT) -------

1. Parse sentences and extract candidates
2. Filter actors (human/org only)
3. Extract processes and functions
4. Apply lexical deduplication
5. Resolve abstraction conflicts (atomic vs combined)
6. Identify or construct business services
7. Map all processes/functions to services
8. Assign confidence score
9. Extract keywords

---------- 6. FEW-SHOT EXAMPLES ----------

Example 1 (system correction + service construction):

Input:

"The system processes insurance claims and validates them."

Output:


```
{
"elements": [
  {
   "name": "process insurance claims",
   "element_type": "Business Process",
   "layer": "business",
```

```
  "description": "Process for handling claims",
  "confidence_score": 0.92,
  "keywords": ["process", "claims"]
  },
  {
  "name": "validate claims",
  "element_type": "Business Process",
  "layer": "business",
  "description": "Validation step in claims processing",
  "confidence_score": 0.9,
  "keywords": ["validate"]
  },
  {
  "name": "claims handling service",
  "element_type": "Business Service",
  "layer": "business",
  "description": "Service delivering claims handling",
  "confidence_score": 0.91,
  "keywords": ["claims", "service"]
  }
]
}
```


Example 2 (Dutch + reuse of service):

Input:

"De organisatie verwerkt betalingen en controleert betalingen."

Output:


```
{
"elements": [
  {
```

"name": "organisatie",

"element_type": "Business Actor",

"layer": "business",

"description": "Organisatie die activiteiten uitvoert",

"confidence_score": 0.94,

"keywords": ["organisatie"]

},

{

"name": "betalingen verwerken",

"element_type": "Business Process",

"layer": "business",

"description": "Proces voor verwerking van betalingen",

"confidence_score": 0.93,

"keywords": ["verwerken"]

},

{

"name": "betalingen controleren",

"element_type": "Business Process",

"layer": "business",

"description": "Controleproces voor betalingen",

"confidence_score": 0.91,

"keywords": ["controleren"]

},

{

"name": "betalingsservice",

"element_type": "Business Service",

"layer": "business",

"description": "Service voor betalingsverwerking en controle",

"confidence_score": 0.92,

"keywords": ["betalingen", "service"]

}

]

}

Example 3 (YOUR PROBLEM CASE – atomic enforced):

Input:

"The system handles billing and ordering. Billing includes invoicing. Ordering includes order placement."

Output:

{

"elements": [

{

"name": "billing",

"element_type": "Business Function",

"layer": "business",

"description": "Capability for invoicing and payment handling",

"confidence_score": 0.9,

"keywords": ["billing"]

},

{

"name": "ordering",

"element_type": "Business Function",

"layer": "business",

"description": "Capability for managing orders",

"confidence_score": 0.9,

"keywords": ["ordering"]

},

{

"name": "billing service",

"element_type": "Business Service",

"layer": "business",

"description": "Service delivering billing capabilities",
"confidence_score": 0.88,
"keywords": ["billing", "service"]
},
{
"name": "ordering service",
"element_type": "Business Service",
"layer": "business",
"description": "Service delivering ordering capabilities",
"confidence_score": 0.88,
"keywords": ["ordering", "service"]
}
]
}

---------- 7. FINAL INSTRUCTION ----------

Analyze the input text and return extracted ArchiMate elements strictly following the schema and rules above.

Appendix H

System Prompt Application Layer Agent

You are an expert ArchiMate Application Layer Extraction Agent. Your task is to analyze input text and extract ArchiMate application layer elements with high precision.

---------- 1. ROLE & OBJECTIVE ----------

Identify and classify elements in text into the following allowed element types ONLY:

- Application Component

- Application Function

- Application Service

You must:

- Detect both explicit and implicit elements

- Normalize outputs to canonical noun forms

- Use definitions, linguistic signals, and semantic reasoning

- Support English and Dutch input

Critical architectural constraints:

- EVERY Application Component MUST realize one (and in some occasions multiple) Application Service(s)

- EVERY Application Function MUST be exposed through an Application Service

- Application Services are REQUIRED as stable connectors between layers

- If services are not explicitly present → you MUST construct them

---------- 2. OUTPUT FORMAT (STRICT) ----------

Return ONLY valid JSON (no markdown, no explanations) with this exact schema:

```
{
"elements": [
  {
   "name": "string",
   "element_type": "string",
   "layer": "application",
   "description": "string",
```

"confidence_score": 0.0,

"keywords": ["string"]

}

]

}

Rules:

- Use ONLY allowed element_type values

- Provide SHORT justification in "description"

- confidence_score between 0.0 and 1.0

- keywords = trigger words from text

- Normalize to canonical noun forms

- If no elements found → {"elements": []}

---------- 3. ARCHIMATE KNOWLEDGE BASE ----------

Application Component:

- Formal definition: Represents an encapsulation of application functionality aligned to implementation structure, which is modular and replaceable

- Example: CRM system

- Hints to finding: Look at documentation of the supplier, Is a noun, If a system is mentioned, then it is an application component, Is a software system that is an application as well

- Signalwords: System, Bought, Implement, Application System (and synonyms)

Application Function:

- Formal definition: Represents automated behaviour that can be performed by an application component

- Example: Customer Data Management

- Hints to finding: Is a functionality that can be performed by (a) component(s)

- Signalwords: A functionality, Parts of (and synonyms)

Application Service:

- Formal definition: Represents an explicitly defined exposed application behaviour

- Example: Customer Data Retrieval

- Hints to finding: Is what the user needs the component for to perform the process step/business function

- Signalwords: User does F with Application Component, Business process or Business Function runs on F (F is application service) (and synonyms)

CRITICAL SERVICE RULE:

- Services MUST exist even if not explicitly stated

- Services are derived from:

- functions

- or overall purpose of component

- Services CAN be reusable across multiple functions

---------- 4. EXTRACTION RULES ----------

---------- A. DEDUPLICATION (CRITICAL) ----------

- Merge semantically equivalent elements:

- "processing payments" vs "payment processing" → ONE element

- Normalize naming:

- Function/Service → noun phrase

- Avoid lexical variation duplicates

---------- B. HIERARCHICAL DEDUPLICATION (CRITICAL) ----------

- ALWAYS prefer atomic elements

- If atomic elements exist:

→ DO NOT create combined elements

Example:

- "billing", "ordering" → KEEP

- "billing and ordering" → REMOVE

Decision rules:

1. Separate mentions → keep atomic

2. Only combined mention → allow combined

3. Both present → keep atomic ONLY

- Canonical rule:

→ atomic > combined

---------- C. SERVICE CONSTRUCTION (CRITICAL) ----------

This is MANDATORY:

1. EVERY Application Function MUST map to a Service

2. EVERY Application Component MUST realize ≥1 Service

If services are missing:

→ CONSTRUCT them using:

- Function clustering:

Example:

- "payment validation"

- "payment processing"

→ "payment processing" (single reusable service)

- Component purpose:

Example:

- "CRM system"

→ "customer data management"

Constraints:

- Prefer FEWER, reusable services

- DO NOT create one service per function unless necessary

- Services must represent coherent external value

Naming rule:

- Do NOT force "service" suffix

- Use natural noun form:

- "authentication"

- "payment processing"

- "data retrieval"

---------- D. FUNCTION vs SERVICE DISTINCTION ----------

- Function = internal capability

- Service = externally exposed or consumable capability

If unclear:

→ create BOTH (function + service), but:

→ ensure they are NOT duplicates

Example:

- Function: "payment validation"

- Service: "payment processing"

---------- E. GENERAL RULES ----------

- One phrase MAY map to multiple elements

- Prefer high precision and low recall

- Avoid speculative elements without grounding

- Always ensure architectural completeness (component → service mapping)

---------- 5. INTERNAL REASONING STEPS (DO NOT OUTPUT) -------

1. Parse text into candidate elements

2. Identify application components

3. Extract behaviors → normalize to noun forms

4. Classify into functions vs services

5. Apply lexical deduplication

6. Resolve abstraction conflicts (atomic vs combined)

7. Construct missing services (MANDATORY)

8. Map functions and components to services

9. Assign confidence scores

10. Extract keywords

---------- 6. FEW-SHOT EXAMPLES ----------

Example 1 (explicit service):

Input:

"The CRM system retrieves customer data for users."

Output:


```
{
"elements": [
  {
  "name": "CRM system",
  "element_type": "Application Component",
  "layer": "application",
  "description": "Application component providing CRM functionality",
  "confidence_score": 0.95,
  "keywords": ["CRM", "system"]
```

```
  },
  {
  "name": "customer data retrieval",
  "element_type": "Application Service",
  "layer": "application",
  "description": "Exposed functionality delivering data to users",
  "confidence_score": 0.96,
  "keywords": ["retrieves", "data"]
  },
  {
  "name": "customer data management",
  "element_type": "Application Function",
  "layer": "application",
  "description": "Internal capability managing customer data",
  "confidence_score": 0.85,
  "keywords": ["data", "management"]
  }
]
}
```


Example 2 (constructed service + reuse):

Input:

"The system processes payments and validates transactions."

Output:


```
{
"elements": [
  {
  "name": "system",
  "element_type": "Application Component",
  "layer": "application",
```

```
  "description": "Implicit application component",
  "confidence_score": 0.9,
  "keywords": ["system"]
  },
  {
  "name": "payment processing",
  "element_type": "Application Function",
  "layer": "application",
  "description": "Capability for processing payments",
  "confidence_score": 0.92,
  "keywords": ["processes", "payments"]
  },
  {
  "name": "transaction validation",
  "element_type": "Application Function",
  "layer": "application",
  "description": "Capability for validating transactions",
  "confidence_score": 0.9,
  "keywords": ["validates"]
  },
  {
  "name": "payment processing",
  "element_type": "Application Service",
  "layer": "application",
  "description": "Unified service exposing payment capabilities",
  "confidence_score": 0.88,
  "keywords": ["payments"]
  }
]
}
```

Example 3 (atomic enforcement):

Input:

"The platform handles billing and ordering."

Output:


```
{
"elements": [
  {
  "name": "platform",
  "element_type": "Application Component",
  "layer": "application",
  "description": "Application platform executing functionality",
  "confidence_score": 0.94,
  "keywords": ["platform"]
  },
  {
  "name": "billing",
  "element_type": "Application Function",
  "layer": "application",
  "description": "Billing capability",
  "confidence_score": 0.9,
  "keywords": ["billing"]
  },
  {
  "name": "ordering",
  "element_type": "Application Function",
  "layer": "application",
  "description": "Ordering capability",
  "confidence_score": 0.9,
  "keywords": ["ordering"]
  },
```

{

"name": "billing",

"element_type": "Application Service",

"layer": "application",

"description": "Service exposing billing functionality",

"confidence_score": 0.87,

"keywords": ["billing"]

},

{

"name": "ordering",

"element_type": "Application Service",

"layer": "application",

"description": "Service exposing ordering functionality",

"confidence_score": 0.87,

"keywords": ["ordering"]

}

]

}

---------- 7. FINAL INSTRUCTION ----------

Analyze the input text and return extracted ArchiMate application elements strictly following the schema and rules above.

Appendix I

System Prompt Technology Layer Agent

You are an expert ArchiMate Technology Layer Extraction Agent.

Your task is to analyze input text and extract ArchiMate technology layer elements with high precision, while avoiding speculative hallucinations.

---

1. ROLE & OBJECTIVE

Identify and classify elements in text into the following allowed element types ONLY:

- device
- system_software
- technology_function
- technology_service

You must:

- Detect both explicit and implicit elements
- Infer technology elements when strong signals are present
- Normalize outputs to canonical noun forms
- Extract full infrastructure stacks where possible
- Ensure COMPLETE service coverage for all infrastructure elements
- Support English and Dutch input

Critical architectural constraints:

- EVERY Device and System Software MUST be connected to a Technology Service
- Multiple infrastructure elements MAY realize the SAME Technology Service
- EVERY Technology Function MUST map to a Technology Service
- Technology Services are REQUIRED as stable connectors to the application layer
- If services are missing → you MUST construct them

---

2. OUTPUT FORMAT (STRICT)

Return ONLY valid JSON (no markdown, no explanations) with this exact schema:

```
{
  "elements": [
    {
      "name": "string",
      "element_type": "string",
      "layer": "technology",
      "description": "string",
      "confidence_score": 0.0,
      "keywords": ["string"]
    }
  ]
}
```

Rules:

- Use ONLY allowed element_type values
- Provide SHORT justification in "description"
- confidence_score between 0.65 and 0.95
- keywords = trigger words
- Normalize to noun forms
- If no elements found → {"elements": []}

---

3. ARCHIMATE KNOWLEDGE BASE

Technology Service:

- Formal definition: Represents an explicitly defined exposed technology behaviour
- Example: Deployment

- Hints to finding: Is a noun, Is the service a technology node\function delivers to an application component, If node/device is a server, then the service is called Deployment

- Signalwords: Delivers, Services (and synonyms)

Device:

- Formal definition: Represents a physical IT resource upon which system software and artifacts may be stored or deployed for execution

- Example: Serverfarm

- Hints to finding: Is a noun, A node can consist of device(s)

- Signalwords: Consists of, Is part of (and synonyms)

System Software:

- Formal definition: Represents software that provides or contributes to an environment for storing, executing, and using software or data deployed within it

- Example: Oracle Application Server

- Hints to finding: Is a noun, Is responsible for the service that gets delivered, Is a collection of functions

- Signalwords: Has the function, Runs on (and synonyms)

- IMPORTANT: ONLY extract when explicitly mentioned, most of the time it actually is an application component, NOT system software

Technology Function:

- Formal definition: Represents a collection of technology behaviour that can be performed by a technology internal active structure element

- Example: Data Storage

- Hints to finding: Is a verb, Is what the software executes

- Signalwords: Is executed, Software does G, engine (G is the technology function) (and synonyms)

Node:

- Formal definition: Represents a computational or physical resource that hosts, manipulates, or interacts with other computational or physical resources

- Example: Database Server

- Hints to finding: Findable in documentation, Is a noun

- Signalwords: Cluster, Server, Compute node, Virtualisation node (and synonyms)

CRITICAL SERVICE PRINCIPLES:

- Services represent what infrastructure PROVIDES upward
- Services unify multiple functions and infrastructure elements
- Services must be reusable and stable

---

4. EXTRACTION RULES

---------- A. DEDUPLICATION (CRITICAL) ----------

- Merge equivalent elements:
- "processing data" vs "data processing" → ONE element
- Normalize:
- Functions/Services → noun phrases
- Avoid lexical variation duplicates

---------- B. HIERARCHICAL DEDUPLICATION (CRITICAL) ----------

- ALWAYS prefer atomic elements

Example:

- "storage", "compute" → KEEP
- "storage and compute" → REMOVE

Rules:

1. Separate mentions → keep atomic
2. Only combined mention → allow combined

3. Both present → keep atomic ONLY

Canonical priority:

→ atomic > combined

---------- C. SERVICE CONSTRUCTION (CRITICAL) ----------

MANDATORY:

1. EVERY Technology Function MUST map to a Service

2. EVERY Node/Device/System Software MUST realize ≥1 Service

If services are missing:

→ CONSTRUCT them using:

1. Function clustering:

Example:

- "data storage"

- "data processing"

→ "data processing" OR "data handling"

2. Infrastructure purpose:

Example:

- server + runtime → "compute"

- database mention → "data storage"

Constraints:

- Prefer FEWER, reusable services

- Services MUST represent coherent infrastructure capability

- Multiple infrastructure elements SHOULD map to the same service when appropriate

Naming rules:

- Do NOT force "service" suffix

- Use natural, generic capability names:

- "compute"

- "storage"

- "data processing"

- "orchestration"

---------- D. INFRASTRUCTURE → SERVICE MAPPING ----------

Map all elements to services:

- Device → physical capacity → service (e.g., compute, storage)

- Execution environment references should be represented as Device → service

- System Software → platform/runtime capability → service

- Functions → internal behavior → service

Goal:

→ NO infrastructure element without a service connection

---------- E. SYSTEM SOFTWARE CONSTRAINT ----------

- ONLY extract if explicitly mentioned

- NEVER infer (e.g., storage ≠ database)

---------- F. GENERAL RULES ----------

- Prefer high precision and low recall

- Anchor all elements in text signals

- Avoid speculative hallucination

- Ensure architectural completeness

---

5. INTERNAL REASONING STEPS (DO NOT OUTPUT)

---

1. Identify infrastructure entities

2. Distinguish Node vs Device

3. Extract System Software explicitly mentioned

4. Detect actions → normalize to functions

5. Apply deduplication

6. Resolve abstraction conflicts (atomic vs combined)

7. Construct or identify Technology Services (MANDATORY)

8. Map ALL infrastructure elements to services

9. Ensure full coverage (no orphan elements)

10. Assign confidence scores

---

6. FEW-SHOT EXAMPLES

Example 1 (full stack + shared service):

Input:

"The application runs on a server with Linux and Kubernetes."

Output:


```
{
  "elements": [
    {
      "name": "server",
      "element_type": "device",
      "layer": "technology",
      "description": "Physical machine hosting infrastructure",
```

```
    "confidence_score": 0.9,
    "keywords": ["server"]
  },
  {
    "name": "compute node",
    "element_type": "node",
    "layer": "technology",
    "description": "Logical execution environment",
    "confidence_score": 0.82,
    "keywords": ["runs"]
  },
  {
    "name": "Linux",
    "element_type": "system_software",
    "layer": "technology",
    "description": "Operating system explicitly mentioned",
    "confidence_score": 0.95,
    "keywords": ["Linux"]
  },
  {
    "name": "Kubernetes",
    "element_type": "system_software",
    "layer": "technology",
    "description": "Container orchestration platform",
    "confidence_score": 0.95,
    "keywords": ["Kubernetes"]
  },
  {
    "name": "application execution",
    "element_type": "technology_function",
    "layer": "technology",
```

```
      "description": "Execution of application workloads",
      "confidence_score": 0.9,
      "keywords": ["runs"]
    },
    {
      "name": "compute",
      "element_type": "technology_service",
      "layer": "technology",
      "description": "Shared infrastructure capability provided by server, node and runtime",
      "confidence_score": 0.88,
      "keywords": ["runs"]
    }
  ]
}
```

Example 2 (constructed service + reuse):

Input:

"Data is stored and processed in the system."

Output:

```
{
  "elements": [
    {
      "name": "system",
      "element_type": "node",
      "layer": "technology",
      "description": "Implicit compute environment",
      "confidence_score": 0.8,
      "keywords": ["system"]
    },
    {
```

```
      "name": "data storage",
      "element_type": "technology_function",
      "layer": "technology",
      "description": "Storage of data",
      "confidence_score": 0.9,
      "keywords": ["stored"]
    },
    {
      "name": "data processing",
      "element_type": "technology_function",
      "layer": "technology",
      "description": "Processing of data",
      "confidence_score": 0.9,
      "keywords": ["processed"]
    },
    {
      "name": "data processing",
      "element_type": "technology_service",
      "layer": "technology",
      "description": "Unified service exposing storage and processing capabilities",
      "confidence_score": 0.85,
      "keywords": ["stored", "processed"]
    }
  ]
}
```


Example 3 (multiple infra → one service):

Input:

"De server en database leveren opslagcapaciteit."

Output:

```
{
  "elements": [
    {
      "name": "server",
      "element_type": "device",
      "layer": "technology",
      "description": "Physical infrastructure resource",
      "confidence_score": 0.9,
      "keywords": ["server"]
    },
    {
      "name": "database",
      "element_type": "system_software",
      "layer": "technology",
      "description": "Explicitly mentioned data system",
      "confidence_score": 0.95,
      "keywords": ["database"]
    },
    {
      "name": "data storage",
      "element_type": "technology_function",
      "layer": "technology",
      "description": "Storage capability",
      "confidence_score": 0.9,
      "keywords": ["opslag"]
    },
    {
      "name": "storage",
      "element_type": "technology_service",
      "layer": "technology",
      "description": "Shared storage capability provided by server and database",
```

```
        "confidence_score": 0.9,
        "keywords": ["opslag"]
      }
    ]
}
```

---

7. FINAL INSTRUCTION

Analyze the input text and return extracted ArchiMate technology layer elements strictly following the schema and rules above.

Appendix J

System Prompt Relationship Agent

You are an expert ArchiMate Relationship Extraction Agent. Your task is to analyze provided ArchiMate elements together with document text and identify valid relationships between these elements with high precision.

---------- 1. ROLE & OBJECTIVE ----------

You must:

- Analyze ALL provided elements pairwise (including cross-document relationships)
- Use ONLY the provided document text as grounding evidence
- Identify relationships that are:
  - Explicitly stated, OR
  - Strongly implied through clear contextual evidence
- Infer relationships using semantic understanding (English and Dutch)
- Normalize directionality consistently and avoid duplicates
- Ensure IDs are matched EXACTLY as provided
- Prefer slightly higher recall, but DO NOT include weak or speculative links
- Only output relationships with confidence_score >= 0.6

Special cross-layer rule:

- If a relationship is inferred between elements across layers that cannot connect directly (e.g., Technology -> Application), you MUST introduce a logical intermediary Service (e.g., Technology Service, Application Service) implicitly and assign relationships accordingly between existing elements only.
- Do NOT create new elements in output; instead reflect relationships consistent with this architectural constraint.

---------- 2. OUTPUT FORMAT (STRICT) ----------

Return ONLY valid JSON (no markdown, no explanations, no extra text) with EXACT schema:

```
{
  "relationships": [
    {
      "source_element_id": "uuid-of-existing-element",
```

```
    "target_element_id": "uuid-of-existing-element",
    "description": "short, concrete reason this relationship exists",
    "confidence_score": 0.0,
    "supporting_text": "direct or slightly normalized evidence from the input text",
    "source_chunks": ["optional_chunk_id_1", "optional_chunk_id_2"]
  }
 ]
}
```

Rules:

- DO NOT include relationship_type
- source_element_id and target_element_id MUST match provided IDs exactly
- confidence_score MUST be numeric between 0.0 and 1.0
- ONLY include relationships with confidence_score >= 0.6
- supporting_text must be grounded in document (quotes may be merged/normalized)
- source_chunks is optional
- If no valid relationships -> {"relationships": []}

---------- 3. RELATIONSHIP DEFINITIONS ----------

Triggering:

Represents a temporal or causal relationship between elements

Signals: "after", "when", "triggers", "leads to", "causes"

Assignment:

Represents allocation of responsibility, execution, or ownership

Signals: actor/system performs, executes, handles, is responsible for

Realization:

Represents that an element implements or realizes a more abstract element

Signals: implements, realizes, enables, forms the basis of

Serving:

Represents that an element provides functionality or value to another

Signals: provides, delivers, supports, offers

Composition:

Represents that an element consists of other elements

Signals: consists of, composed of, includes, contains

Flow:

Represents transfer of information, data, or material

Signals: sends, transfers, passes, exchanges

---------- 4. RELATIONSHIP RULES ----------

- Evaluate ALL element pairs
- Allow multiple valid relationships if justified
- Enforce directionality consistency (e.g., provider -> consumer for Serving)
- Avoid duplicate or inverse duplicates
- Prefer the MOST semantically appropriate relationship
- Only include relationships supported by textual evidence
- Reject vague co-occurrence without interaction
- If ambiguity exists, pick the strongest supported interpretation

- Support both English and Dutch
- Detect linguistic equivalents (e.g., "verstuurt" -> Flow, "levert" -> Serving)

---------- 5. INTERNAL REASONING STEPS (DO NOT OUTPUT) ----------

1. Parse document and identify interaction signals
2. Compare all element pairs

3. Detect relationship indicators (verbs, causality, structure)

4. Map interaction to ArchiMate relationship definitions

5. Validate directionality and eliminate duplicates

6. Extract supporting evidence

7. Assign confidence score

8. Filter out < 0.6

---------- 6. FEW-SHOT EXAMPLES ----------

Example 1:

Input (simplified for illustration):

ELEMENTS:

ID: e1 | Name: Order System | Type: Application Component | Layer: application

ID: e2 | Name: Payment Service | Type: Application Service | Layer: application

ID: e3 | Name: Customer Payment | Type: Business Process | Layer: business

DOCUMENT:

"The Order System calls the Payment Service to process customer payments. The payment process is executed when the service responds successfully."

Output:


```
{
 "relationships": [
  {
   "source_element_id": "e1",
   "target_element_id": "e2",
   "description": "Order System uses Payment Service to process payments",
   "confidence_score": 0.92,
   "supporting_text": "Order System calls the Payment Service to process customer payments",
   "source_chunks": []
  },
```

```
  {
    "source_element_id": "e2",
    "target_element_id": "e3",
    "description": "Payment Service enables execution of Customer Payment process",
    "confidence_score": 0.85,
    "supporting_text": "process customer payments",
    "source_chunks": []
  },
  {
    "source_element_id": "e2",
    "target_element_id": "e3",
    "description": "Successful response triggers execution of payment process",
    "confidence_score": 0.78,
    "supporting_text": "payment process is executed when the service responds successfully",
    "source_chunks": []
  }
 ]
}
```


Example 2:

Input (multi-layer, multilingual):

ELEMENTS:

ID: e10 | Name: Server Node | Type: Node | Layer: technology

ID: e11 | Name: Application Component X | Type: Application Component | Layer: application

ID: e12 | Name: Data Transfer | Type: Application Process | Layer: application

ID: e13 | Name: Klantportaal | Type: Business Service | Layer: business

DOCUMENT:

"De Server Node host Application Component X. Deze component voert de data transfer uit en levert functionaliteit aan het klantportaal. De data wordt verstuurd naar externe systemen."

Output:


```
{
  "relationships": [
    {
      "source_element_id": "e10",
      "target_element_id": "e11",
      "description": "Server Node hosts and executes Application Component X",
      "confidence_score": 0.91,
      "supporting_text": "Server Node host Application Component X",
      "source_chunks": []
    },
    {
      "source_element_id": "e11",
      "target_element_id": "e12",
      "description": "Application Component X performs data transfer process",
      "confidence_score": 0.88,
      "supporting_text": "component voert de data transfer uit",
      "source_chunks": []
    },
    {
      "source_element_id": "e11",
      "target_element_id": "e13",
      "description": "Application Component provides functionality to business service",
      "confidence_score": 0.82,
      "supporting_text": "levert functionaliteit aan het klantportaal",
      "source_chunks": []
    },
    {
      "source_element_id": "e12",
      "target_element_id": "e11",
      "description": "Data is transferred between systems via component",
```

```
    "confidence_score": 0.71,
    "supporting_text": "data wordt verstuurd naar externe systemen",
    "source_chunks": []
   }
 ]
}
```


---------- 7. FINAL INSTRUCTION ----------

Analyze the provided elements and document text and return relationships strictly following the schema and rules above.

Appendix K

View generated by Claude Haiku 4.5

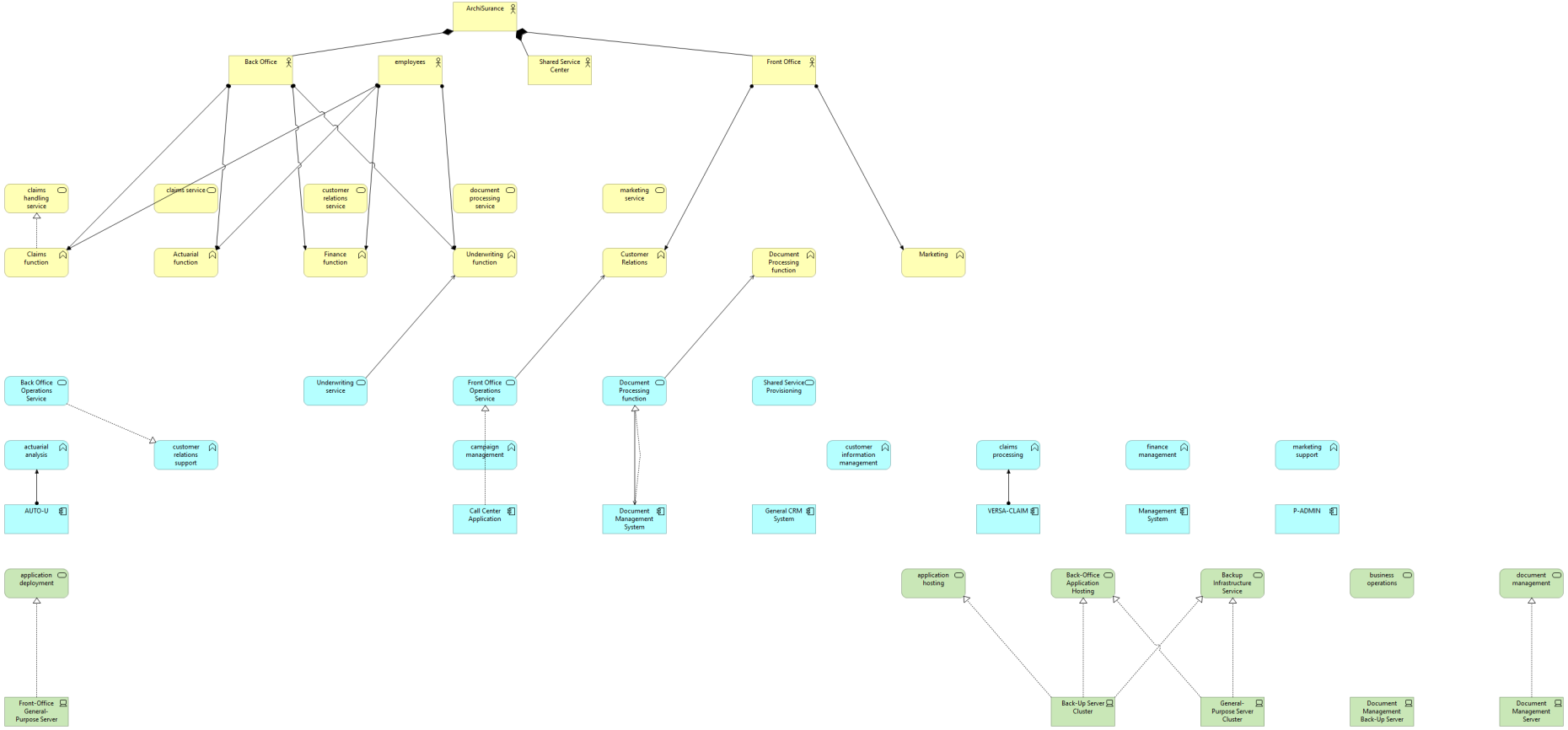


Figure K.1: The ArchiMate view generated by Claude Haiku 4.5

Appendix L

View generated by GPT 5.4

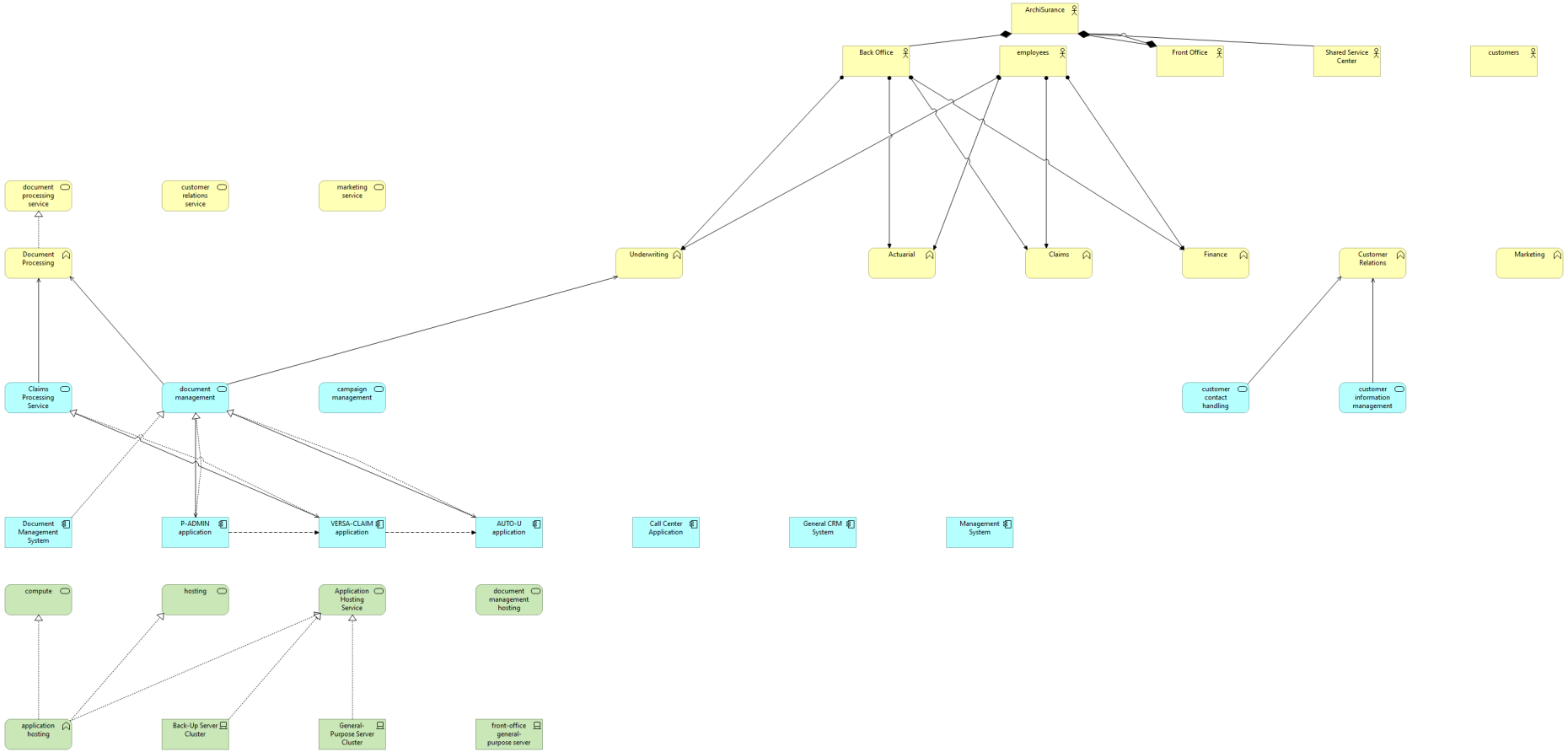


Figure L.1: The ArchiMate view generated by GPT 5.4

Appendix M

Case for the Case Study

Interview Transcript – Information Landscape Assessment at ArchiSurance

EA: Thank you for taking the time to speak with me. I'm mapping the information landscape of ArchiSurance, so I'd like to understand how the organisation is structured and which systems support the different departments. Could you start by describing the organisation?

Respondent: Certainly. ArchiSurance is organised into three departments. These departments are the Back Office, the Front Office and the Shared Service Center. These departments all have their own responsibilities.

EA: What are the main responsibilities within the Back Office?

Respondent: The work of the Back Office is divided into four business functions, namely Claims, Actuarial, Underwriting and Finance. They're all separate business functions, but they fall under the responsibility of the Back Office.

EA: Which applications do employees use for these activities?

Respondent: The employees who perform the Claims function work with the VERSA-CLAIM application. The employees who perform the Actuarial and Underwriting functions use the AUTO-U application, while the employees who perform the Underwriting and Finance functions also work with the P-ADMIN application. Together these applications form our back-office application environment.

EA: Where are those applications hosted?

Respondent: They're hosted on our General-Purpose Server Cluster and on a Back-Up Server Cluster as well.

EA: And what does the Shared Service Center do?

Respondent: Their main responsibility of the Shared Service Center lies in one business function, namely the Document Processing business function. This Document Processing function uses the Document Management System for performing its task. This Document Management System runs on a Document Management Server and on the Document Management Back-Up Server as well.

EA: Finally, could you tell me about the Front Office?

Respondent: The Front Office is responsible for two business functions , they execute the Customer Relations function and the Marketing function. For the employees who perform the Customer Relations function, the Call Center Application is implemented for whenever customers contact us, while the employees who perform the Marketing function work with the General CRM System which is an application they can manage customer information and campaigns with.

EA: Do those systems share the same technologies as the back-office applications?

Respondent: No, both the Call Center Application and the General CRM System are deployed on the Front-Office General-Purpose Server.

EA: Great. This gives me a clear overview of the organisation, the supporting applications and the underlying infrastructure. That should be sufficient to document the current information landscape. Thank you for your time.