Universiteit Leiden

ICT in Business and the Public Sector

Generating process models from textual requirements using transformer based natural language processing

Name: Pepijn Griffioen
Student-no: s1673998

Date: 09/07/2022

1st supervisor: Dr. G.J. Ramackers
2nd supervisor: Dr. P.W.H. van der Putten

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Generating process models from textual requirements using transformer based natural language processing

Pepijn Griffioen

Leiden Institute of Advanced Computer Science
P.O. Box 9512, 2300 RA Leiden, The Netherlands

July 9, 2022

Abstract

Requirements to build a system are created and documented using different techniques and models, the goal of these is to enable clear communication about the requirements. One of these models is the UML activity diagram, which depicts the behaviour of a process in a system in a graphical manner. The creation of such models is time-consuming and error-prone. To overcome this problem we present a NLP pipeline to transform process descriptions into activity models. This pipeline is a combination of novel transformer-based NLP models and rules to transform a description into an activity model. We have compared our results with a BPMN baseline set of models and see that our model creates actions and conditions with more context. Moreover, with our combination of semantic role labelling, coreference and natural language inference we can combine different conditions with more certainty than existing approaches. Furthermore, we have implemented a UML activity metadata model to store models and implemented it with a set of APIs to integrate it within the Prose to Prototype environment. We have also collaborated with a low-code vendor to explore the opportunities to further validate our approach in a real-world context. Based on our research and development we have provided an environment that creates editable activity models from a requirements text. This solution decreases development time and helps the modeller create better solutions.

Acknowledgements

I would like to acknowledge some people who, without I would not have finished this thesis. First of all, I would like to thank my first supervisor Dr. Ramackers, who I had the pleasure to get to know and meet. We had endless discussions on the different topics that we would like to address and it was inspiring to talk about the improvements to the P2P system. Also, the meetings that he managed to set up and the people I got to meet was an amazing experience. Guus was also good at keeping me challenged, which was sometimes frustrating, but I got to learn a lot. Next to that, I would like to thank my second supervisor Dr. van der Putten. Peter kept me focused and gave me great feedback. Also, the collaboration he made possible with Pega was a cool experience! Another professor who was of great value is Dr. Chaudron, who gave me feedback, sent some foundational literature for this thesis and asked the right questions to keep me on track.

As for my fellow students, I would like to thank Martijn Schouten for being the person to discuss NLP ideas with and to improve the P2P system. Bram van Aggelen who created the visual appearance of the UML editor and Willem-Pieter van Vlokhoven who build the APIs.

I would also like to thank my family and friends who have been supportive and of great help to make sure I finished this thesis. I would like to especially thank my girlfriend Nadine who supported me and was the person I could always talk to. Overall I am happy with the result and look forward to the future.

Pepijn Griffioen

Contents

Chapter 1

Introduction

Enterprise system development is the process of developing IT systems within the context of an enterprise. The process can be very complex because there are multiple aspects involved such as the business environment, the number of stakeholders and the IT landscape. The systems that are developed have a certain business focus, to overcome a problem or task within the environment of the enterprise.

Several stakeholders are involved in the development process, which are business specialists, business experts, UML experts and developers. The different roles and backgrounds of the stakeholders make the development process a multi-disciplinary effort[1]. The involvement of the different employees is necessary to tailor the system to their needs.

The collaboration between these employees introduces a difference in background, that presents a knowledge gap between them. To overcome this difference in backgrounds, graphical models are used to allow each of the employees to understand the system that is developed. Next to models themselves there are also approaches based on these models such as code generation or model-driven software, to create applications. The mentioned approaches contribute to the increase of importance that graphical design models have on the development process. The model we consider in this research is the UML activity diagram, which shows the functionality of a process, the different actors and the relations between the functionality and the actors. Such models allow the different employees, regardless of their background, to understand the system.

Models such as the activity diagram are a result of a requirements engineering process. The requirements engineering process can vary depending on the system that is being developed. Often the starting activities of this process are the same, these are the elicitation and analysis activi-

ties. The elicitation activity is focused on collecting the requirements and in the analysis activity the requirements will be analysed, both activities can have a set of written requirements, a graphical model or even a prototype application as a result. The requirements will then be discussed and changed if needed.

In an enterprise context, the main area of interest is business processes. Activity models can model such business processes. The models enable the business user to understand the processes. In this research, the focus is on models that are used in an enterprise context. The development of such models and requirements can be a complex task. To support such development several approaches have been created, one often used area of techniques is Natural Language Processing. The area consists of several models and algorithms that are able to extract information from natural language, such as textual requirements documents.

1.1 Problem statement

The requirements for a system are beneficial for communication but are not a silver bullet for system development. To create a requirement several steps are taken, in most cases, it comes down to writing and interpreting what is needed for a system. This process is often done manually by analysts. The manual interpretation of text and models can easily introduce mistakes, it takes quite some time and it is complex to change one of the requirements concerning their dependencies. These obstacles combined create a significant financial burden in the overall process. Besides the obstacles, a well-known IT phenomenon is the growth of costs for removing defects. The further a development process progresses the more expensive the removal of a defect is compared to an earlier stage[2]. Therefore the whole process is time-consuming, error-prone and expensive. Nevertheless, the textual requirements and models should not be removed, because they are important for the overall process. Especially in the communication facet of the development process because it allows the different stakeholders to understand the system.

As earlier mentioned Natural Language Processing (NLP) is a technique to support the process of requirements engineering. NLP is a subfield of machine learning, where the main objective is to process natural language. It is currently being applied in several tasks and processes in our society such as chatbots, sentiment analysis and translation applications. These applications are powered with one or more NLP tools and/or models. Recently the field of NLP has seen a significant rise in attention

as a result of the introduction of transformer-based models. The 'transformer' models performed better than a lot of state-of-the-art models at the time.

This performance increase is also interesting for the requirements engineering processes because a large workload in requirements engineering is from text documents. A community that is doing research into the NLP and Requirements engineering area is called NLP4RE*. The research area is at the moment of writing an active and thriving area within the requirements engineering domain[3]. The introduction of new NLP models and current interest in the community shows high promises for new developments, therefore we will investigate how we can use NLP to support the requirements engineering process. Specifically, the research will focus on the use of transformer models to generate activity models, thus our research question is as follows:

> **Main research question:** How can Natural Language Processing be used for the improvement of the iterative elicitation and analysis activities in the requirements engineering process, with a focus on activity models?

The objective of this research is to enable closer collaboration between stakeholders in the system development process through the use of activity models. To find a solution for the main research question the research approach is presented in Section 1.3.

1.2 Research objective

To give the research question a clear objective, we present our research objectives:

- *RO_0: Explore automated approaches to generate activity models from earlier research*
- *RO_1: Build a prototype to transform a requirements text into an activity model using transformer based models.*

In the next chapters, we will go into these research objectives. In the next section, we will define how we aim to approach these objectives in our research.

*https://nlp4re.github.io/2022/

1.3 Research approach

To find a solution to the problem statement, we propose a Design Science research approach. The methodology of Design Science (DS), is a well-known research method in Computer Science[4]. In DS innovative artefacts are developed to gain an understanding of a specific problem, therefore this research will develop a prototype as a solution for the problem statement. To support the prototype with a scientific background, a literature review will also be conducted.

The literature review will focus on the following research questions:

RQ1: What processes and tools with activity models as output are available in literature?

RQ2: What intermediary models and approaches are used for the transformation of requirements into an activity diagram?

The objective of the research questions is to develop a foundation in literature for the development of the prototype. The foundation will make sure the prototype is rooted in literature. In the next section on the methodology, we will further specify how we plan to develop our system following the DS methodology.

1.4 Methodology

The design science approach in this thesis is based on the framework of Peffers et al. that defines a practical framework, as an extension of the design science framework of Hevner et al.[4, 5]. The methodology of Peffers et al. consists of six components: problem identification and motivation, the objective of the solution, design and development, demonstration, evaluation, and communication. These components are reflected in our research. The first two components are discussed in this Chapter, while the development and design component will be discussed in Chapter 3 and 4. The demonstration and evaluation components are part of the results which are discussed in Chapter 5, and the communication component is discussed in the Discussion in Chapter 6.

With this research approach, the prototype is developed in the light of Design Science and rooted in existing literature. Potentially posing a solid prototype for the improvement of the requirements engineering process.

1.5 Academic contribution

In prior work, we have seen different approaches to transforming requirements documents into design models, such as BPMN and Activity models. We will reuse some of their contributions such as keyword-based identification and using agent, verb and object parts of the text to construct our models. We will contribute to the field with the following contributions:

1. Use novel NLP models that are based on the transformer-based architecture.
2. Show a novel combination of NLP tasks to identify, compare and combine conditional structures within the text. Namely the tasks of coreference and natural language inference.
3. Present an Activity metadata class model for the storage of activity models.
4. Create a pipeline for the transformation of requirements texts into activity models. That can be reused, upgraded and extended in the future.

1.6 Overview

The thesis is structured into several chapters and sections. Chapter 2 describes the different transformation approaches and datasets from related literature. Chapter 3 explains how we use the different NLP models to process several tasks to transform the text into an activity model. In Chapter 4 the system design is discussed, which defines how all the different components interact with one another. Then Chapter 5 demonstrates the transformation of several examples from a requirements text into a model and qualitatively compares them with earlier research. We discuss the final findings in Chapter 6 which is structured as a discussion, limitation and future work. Finally, Chapter 7 concludes the research, with the key takeaways.

Chapter 2

Background

The research we conduct is an investigation into several areas, such as Natural Language Processing, UML models and the transformation of natural language into activity models. For the prototype that we develop the input consists of requirements texts and our output is an activity model. To process an input text, we need to extract information from the text, for that we make use of NLP models. Then to create an activity model we require knowledge of the UML activity model domain. In this chapter, we will investigate the different domains and give a theoretical background for them. Besides the theoretical background, we will also investigate related literature on the approaches to transforming text to UML models and give an overview of the available datasets.

The chapter is divided in the following sections, Section 2.1 describes the foundations of Activity models, Section 2.2 explains the theory on the NLP models that we use. Then Section 2.3 describes the different approaches for transformations from earlier work and Section 2.4 gives an overview of the different available datasets to validate our transformation approach.

2.1 Activity models

In the following section, we explain the theory of activity models. We will discuss the components of activity models, the use cases for activity models and the difference with similar models such as BPMN models.

2.1.1 UML Activity Models

Unified Modelling Language (UML) is a modelling standard that is adopted and still managed by the Object Management Group. The Object Management Group refers to it as: ”a graphical language for visualizing, specifying, constructing and documenting the artefacts of distributed object systems”[6]. It is a standardisation that is widely used in software development. The UML models and visualisations can support the system development and requirements engineering process. UML is also used in code generation tasks, where models are converted to code skeletons. There are also model-driven engineering (MDE) applications that use the UML model to completely generate the application, such as low code or no-code platforms. UML models can be categorised into two areas: structural models and behaviour models. Both areas have several different diagrams, which are used depending on the use case.

In this research, we aim to generate activity models. Activity models are specified as behavioural diagrams. Such models are used to model the behaviour of a system. An example of an activity model can be found in Figure 2.1. The figure is from Booch et al., where it is used as an example to explain activity diagrams[7].

The Activity model and its components are extensively explained in the article of Booch et al. [7], to give a summarization of the components we included Table 2.1. The table summarises the components of the activity model, the table is created by Bastos et al.[8].

Booch et al. mention two typical use cases for activity diagram modelling: a workflow or an operation[7]. Workflows are often used for business processes and operations can define computations, such as computational steps. In this research, the goal is to improve the system development process. In system development, business processes are of interest, therefore activity models in the form of workflows are used in this research.

2.1.2 Other process models

As activity models describe the behaviour for a certain process, there are several other approaches to describing certain behaviour. One modelling approach that we also look into in this research is Business Process Model Notation (BPMN), these models are similar to activity models[9]. Nevertheless, there are differences, which are mostly in the notation of certain nodes and certain guidelines or restrictions on how to write such models and their actions.

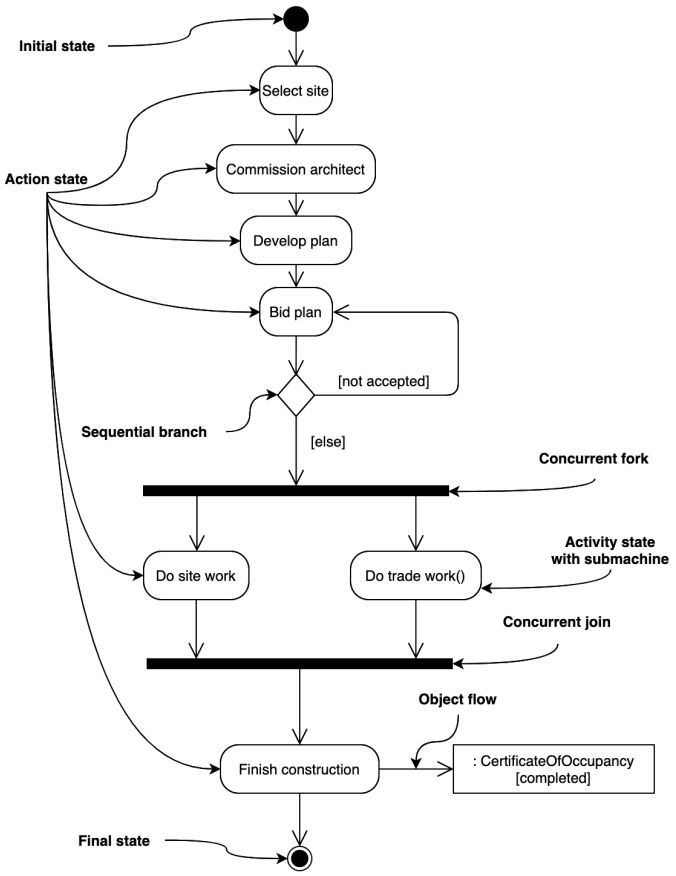

Figure 2.1: *Activity Diagram example[7]*

Concept	Details
Activity	Ongoing non-atomic execution within a state machine. Activities result in some action.
Action	An executable atomic computation that results in a change in the state of the system or the return of a value.
Action State	A state represents the execution of an atomic action, typically the invocation of an operation.
Activity State	It is a composite, whose flow of control is made up of other activity states and action states. Activity states are not atomic meaning that they may be interrupted. Activity states can be further decomposed, their activity being represented by another Activity Diagram.
Transition	Represents the flow of control between two activities. Transition shows the path from one action or activity state to the next action or activity state.
Object Flow	Represent an object involved in a flow of control associated with an activity diagram.
Object State	A condition or situation during the life of the object during which it satisfies some condition, performs, some activity, or waits for some event.
Swimlane	A partition for organising responsibilities for activities. Swimlane does not have a fixed meaning, but they often correspond to organisational units in a business model.

Table 2.1: *Activity diagram concepts[8]*

To demonstrate the similarity between BPMN models and Activity models we present in Figure 2.2 a BPMN example that models a business process. In Figure 2.3 we show an example of an Activity model that models the same process as in Figure 2.2.

Another type of process model can be found in low-code platforms. Most of these platforms have proprietary data models that are often not interchangeable with other models or languages. This is problematic in the case that one would like to migrate from one vendor to another. In some cases, the models do have similarities with activity models. Another common problem with these platforms is the extensibility because the platforms are limited to the functionalities that they offer. In the case that there is a better functionality on another platform they can not just

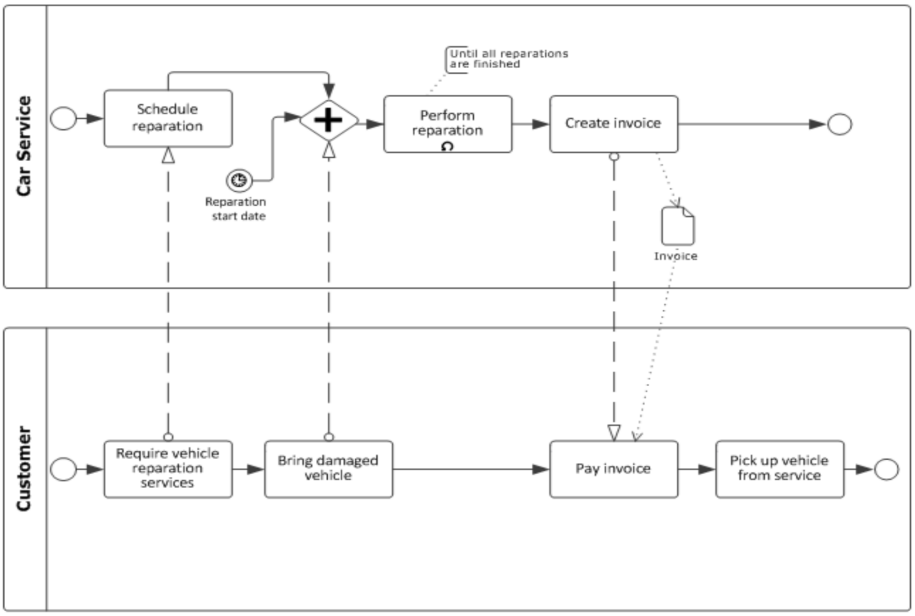

Figure 2.2: *Representation of a business process in BPMN[9]*

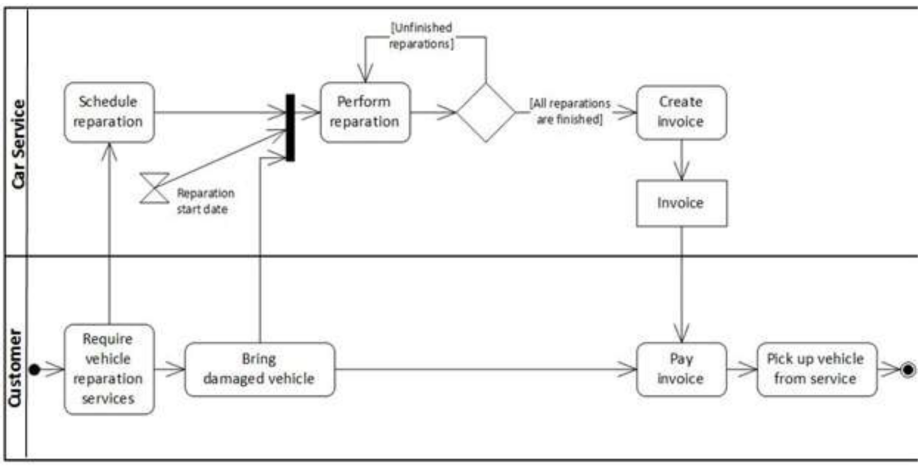

Figure 2.3: *Representation of a business process in a Activity model[9]*

integrate it, because they are limited to what the vendor offers[10]. Nevertheless, vendors need to make implementation decisions therefore in some cases the modelling standards such as UML might be too limited for what they try to achieve. Thus it is a balance between implementing standards and building suitable solutions for the customer. Still, these platforms might be interesting in the light of our solution and as a comparison.

2.2 Natural Language Processing methods

For the transformation of natural language texts into Activity models we will make use of NLP tasks. In this section, we will discuss the several NLP models and tasks, that are used within our solution.

To give an idea of what the NLP tasks are used for we briefly discuss them here. The semantic role labelling task is used to construct the different actions for the activity model. The coreference resolution is used to identify certain entities and find references between the different parts of sentences and conditions. Finally, the natural language inference is used for the comparison of the conditions that might be contradicting one another. We will discuss the theory of each NLP task in the following sections to give insight into what is technically happening.

2.2.1 Semantic Role Labeling

Semantic Role Labeling (SRL) is a Natural Language Processing task, where the objective is to model the predicate-argument structure of a sentence. The result can be described as an answer to "who did what to whom", "when", "where" and more[11]. The information can be used for question answering and information extraction[12].

For the labelling of semantic roles, there are multiple annotation schemes. The schemes from FrameNet[13] and PropBank[14] are considered the main ones. At the moment of writing PropBank is being used for the performance evaluation of the SRL task, with the OntoNotes 5.0 dataset*. Therefore we will further consider the PropBank annotation schema. The PropBank schema is verb sense oriented, which means for every sense of the verb there is a set of role descriptions. The core roles within the sense have general names, such as Arg0, Arg1, Arg2 and Arg3. Each role can differ depending on the verb sense. Nevertheless, the role of Arg0 as the proto-agent and Arg1 as the proto-patient are in most cases the same. The

*https://catalog.ldc.upenn.edu/LDC2013T19

proto-agent is the agent executing the action and the proto-patient is the receiving end of the action. There are also adjunct roles which describe the modifier on the role and can have meanings such as location, temporal, direction and more. An example of a frame for the word 'reserve' is given in Example (1).

(1) Roleset for verb 'reserve'
- **Arg0**: reserver
- **Arg1**: thing reserved
- **Arg2**: benefactive
- **Arg3**: secondary attribute

The roleset in Example (1) shows that Arg0 en Arg1 are fulfilling the proto-agent en proto-patient roles. The roleset is a set of roles for a certain verb sense, in example (2) we present an annotated sentence with this verb.

(2) [The storehouse] has [reserved] [the items on the list].
ARG0 VERB ARG1

As shown in these examples the task of Semantic Role Labeling can identify roles within a sentence. With the use of these roles, we can understand what a sentence means as it can give a sense to a sentence. Also, we can extract the structure from the sentence. In Chapter 3 we will present how we use the SRL task to extract information out of the text to be processed.

2.2.2 Coreference

The task of coreference is to identify mentions in a text that refer to the same entity. These mentions can have several forms, some of the most well-known are personal pronouns, names and demonstrative pronouns. Coreference tries to predict all of these mentions and find the entity they refer to in the text. In the case that a combination of references is found, a set of references can be formed, this is called a cluster or chain. One of the important differences from other NLP tasks is that the coreference task considers all sentences instead of one or a subset of sentences. In example (3) we show an example with an underlining of a cluster of references.

(3) A customer brings in a defective computer and the CRS checks the defect and hands out a repair cost calculation back. If the customer decides that the costs are acceptable, the process continues, otherwise she takes her computer home unrepaired.

As can be seen in example (3) there are two clusters to the same entity. The first cluster refers to the customer and will look as follows: [“a customer”,“the customer”,“she”,“her”]. Next to the customer cluster, there is also a computer cluster: [“a defective computer”,“her computer”]. Based on these clusters it is possible to find connections through the text and make decisions based on other extracted information. In Chapter 3 we demonstrate how we make use of these references and clusters.

2.2.3 Natural language inference

Natural language inference (NLI) is the task of determining what the entailment relation is between two sentences. There are three categories that a sentence can be classified as: entailment, contradiction or undetermined. The task is also known as recognising textual entailment. The task itself is important for multiple aspects for the analysis of complete texts and as input for other NLP tasks. In the following examples, we show two sets of sentences.

(4) Sentences with the label contradiction[15]
   a. Met my first girlfriend that way.
   b. I didn’t meet my first girlfriend until later.

(5) Sentence with the label entailment[16].
   a. A soccer game with multiple males playing.
   b. Some men are playing a sport

As can be seen from the examples (4) and (5) the sentences might not be a clear contradiction or entailment. Nevertheless, the goal of the NLI task is to be able to handle such complex situations. We will be using the NLI task for our solution, the way we make use of this NLP task is described in Chapter 3.

2.2.4 Transformer models

Transformer models are a type of model that have been introduced in the last five years. These models have drastically improved the performance in several NLP tasks. The transformer architecture was introduced by Vaswani et al., it makes use of an encoder-decoder structure and attention mechanisms [17]. Most of the improvements are contributed by the attention mechanisms. That provides for each input token (word) a weight of how important it is when considering another token in the model. This

mechanism enables the model to use the context of the whole text sequence, instead of only the text that has been seen just before. With this contextual awareness, the architecture is very good at using the context of a text.

There are various models available with the transformer architecture and various tasks that the models excel in. The most well-known model is the Bidirectional Encoder Representations from Transformers (BERT), which has proven performance gains on several contextual tasks[18]. The NLP tasks that we mentioned in the previous sections have also seen performance increases, because of the transformer architecture. The models we use in our system will also be transformer-based. In Section 4.3.1 we further elaborate on the NLP platform that we use to implement our models and which models we use.

2.3 Related approaches

Natural language is a complex thing, hard to understand and can be very ambiguous. Nevertheless, it is the way almost everyone communicates with one another in a written or spoken form. There have been many approaches proposed in several domains that aim to understand natural language. Within the domain of transforming requirements texts into reusable activity models, this is also the case. Several contributions in literature propose a way of transforming structured or unstructured texts into a behavioural model.

In this section, we will consider approaches to transform the text into activity or business process models. We chose to include business process models because there are limited research efforts in the transformation space with activity models as an output. Similar to Bellan et al. [19], we give an overview of all the related approaches that we will discuss and their characteristics in Table 2.2. Next to the overview, Bellan et al. have divided the literature into two main transformation approaches, the direct transformation and transformation with an intermediary model. We also make this division in our models, because each approach has its' benefits and downsides.

The main advantage of a direct approach is that you can completely customise the transformation process for the requirements documents. On the contrary that is also a disadvantage, because the direct transformation is not able to create general solutions. In the case that the context changes the approach will perform badly.

For the approach with an intermediary model there is often an iden-

Author	Input	Transform	Approach	Output
van der Aa [20]	Unstructured text	Direct	Rules and templates	DECLARE
Han [21]	Unstructured text	Direct	Artificial Neural Network	BPMN
Sharma [22]	Unstructured text	Intermediary	Rules and frames	Activity & Sequence
Yue [23]	Use case models	Intermediary	Rules and templates	Activity, Sequence & Class
Friedrich [24]	Unstructured text	Intermediary	Rules	BPMN
Sawant [25]	Use case descriptions	Intermediary	Templates	BPMN
Honkisz [26]	Unstructured text	Intermediary	Rules	BPMN

Table 2.2: *Overview of all related transformation approaches.*

tification step and a model building step, that both make use of an intermediary model. The separation of the two steps enables the approach to be easier to extend and the intermediary model requires the two steps to conform to the requirements of the intermediary model. Therefore the advantage of the intermediary model is that in most cases it is more generalised and can move between different contexts. Nevertheless, this does still depend on the implementation and how specific it is to that domain.

Furthermore, in this section, we will present some of the related approaches. They are divided into two transformation types and we will start with the direct transformation.

2.3.1 Direct transformation

A direct transformation approach is an approach where the text is directly transformed into an activity or business process model. The parts of the text that represents parts of processes are transformed into certain actions, decision points, gateways or repetitions.

In the work of van der Aa et al., the authors define an approach to extract declarative process models from natural language. The process models are extracted through a tailored natural language pipeline, which

can identify activities and their relations. The pipeline makes use of part-of-speech tagging and dependency grammar to identify, extract and combine all parts for the process model. For the evaluation, van der Aa et al. made use of 103 pairs of constraint descriptions and declarative constraints. Their model achieved a recall of 0.72 and precision of 0.77, which results in an F1 score of 0.74[20].

Han et al. make use of an Ordered Neurons Long Short-Term Memory (ON-LSTM) model which is trained on Process Definition Documents, which are structured process descriptions. The ON-LSTM can extract a contextual structure from the description and with a learned semantic representation is able to generate a business process and a business process model[21]. The generated diagrams only have an average similarity of only 32% with the taken gold standard, overall it can correctly generate 57% of the edges and 75% of the nodes[21].

2.3.2 Transformation with an intermediary model

The approach with an intermediary model means that there is some model used as a way to temporarily store data, for later reuse. The type of intermediary model does not matter, but it differs from a direct approach.

Sharma et al. present an approach to transform unstructured requirements descriptions automatically into UML activity and sequence diagrams. To transform these texts an intermediary model called frames is filled. The frames are filled using grammatical knowledge patterns and lexical and syntactic analysis using the Stanford Tagger[27] and Parser[28]. There are two limitations, the first is an assumption that there is no redundancy and ambiguity in the written scenario. The second limitation is the requirement that the sequence of occurrence of the actions in the text, is the way the actions should follow one another. Thus making it not possible to refer back or link back to an earlier set of actions.[22].

aToucan is the tool developed in the research of Yue et al., it can automatically transform use case models to UML analysis models such as class, sequence and activity models. The use cases are modelled following the Restricted Use Case Modelling, which enables the use cases to be loaded into an intermediary model. Then it is converted to a metamodel and formalised using the Stanford Parser. Through a transformation of a set of rules a UML model is generated[23]. Unfortunately, the input is limited to structured use cases.

The approach that is mostly used as a baseline by other research is the work from Friedrich et al. [24]. They propose a solution which makes use

of the Stanford CoreNLP library, an algorithm for anaphora resolution and keywords to identify certain gateway types. Furthermore, WordNet and VerbNet are used to extend the keywords. They validate their approach using a dataset of 47 combinations of descriptions and process model diagrams. The models that Friedrich et al. have created, have a 77% similarity with the testing dataset.

Sawant et al. generate an ontology and business process model from use case descriptions. Their approach is an extension of earlier work by Goncalves et al.[29]. Sawant et al. make use of a rule-based approach to specify their world model and use coreference resolution to combine all the entities. In a final step, they build a business process model and extract an ontology which can be used for further analysis. The results are evaluated using a private dataset of 123 use case descriptions[25].

Honkisz et al. create a business process model from texts. The approach extracts Subject-Verb-Object (SVO) triplets, then the actors are extracted and all triplets are clustered together. Followed by a keyword-based extraction, certain gateways are extracted. All the information is saved into an intermediary model, which is used to build a BPMN diagram. A limitation of this paper noted by Bellan et al.[19] is that the results are not thoroughly evaluated[26].

To conclude this section and answer the research questions from Section 1.3 there are several transformation approaches proposed in the existing literature. Business process models are more common to be generated as output than activity models. To power the transformation approach there are several NLP tools and models available, in most cases, the Stanford Parser and Tagger have been used. The most common used NLP tasks are dependency parsing and coreference resolution. There are several different approaches possible with an intermediary model or direct transformation. The most used approach is an intermediary model, an advantage of this approach is the ability to understand where certain aspects of the activity model come from.

2.4 Datasets

Datasets are often used in research to create, test or validate hypotheses. Thus it can support the validity of research that is executed. Unfortunately in the area of requirements engineering the number of datasets is very limited. In Table 2.3 we present an overview of the different requirements datasets. The number of documents specifies the number of documents that are part of the dataset. Labelled specifies if the requirements text is

Table 2.3: *Overview of the different datasets.*

Name	Number of documents	Labelled	Label technique
FN-RE [30]	34	Yes	FrameNet frames [13]
PURE[31]	79	Partially	Self defined XSD format
Process descriptions and BPMN models [32]	47	No	-
PET [33]	47	Yes	BPMN artefacts

labelled and the labelling technique specifies the type of labelling. For the PURE example, the partial label indicates that only 12 of the documents have been labelled. In the case of the Process descriptions and BPMN models from Friedrich et al., there is nothing labelled, instead, a set of corresponding BPMN models is provided with each text. In the next part of this section, we will discuss the different datasets.

Semantic parsing is the technique to add semantic structure to text, but to do this a corpus must be used. Such corpora are missing in the current landscape of NLP4RE. Alhoshan et al. try to fill this gap with the creation of a corpus of requirements documents. FN-RE is the name of the corpus and it is based on the semantic frames in FrameNet[13]. The requirement statements were manually labelled by two annotators, who have an average agreement of 72.85 by f-score, therefore the annotations in the corpus are reliable. The corpus is openly available to support research purposes[30].

Publicly available datasets are not common in the NLP4RE landscape. Ferrari et al. acknowledge this fact in their paper, to overcome this problem they have created a collection of requirements documents under the name of the Public Requirements dataset (PURE). The collection consists of 79 natural language requirements documents collected from the web. In their research they compare the requirements documents with the Brown dataset [34], to show the peculiarities of requirements jargon such as restricted vocabulary and long sentences. Furthermore, the authors have transformed 12 documents in the dataset into a reusable XML format and hope other RE researchers will further annotate the dataset for their own downstream tasks[31].

Friedrich et al. propose an extensive approach to transforming natu-

ral language into business process models[24]. In their approach to validate these models, they have collected 47 process descriptions in combination with a "correct" business process model and their generated process model, the process descriptions are not annotated[24]. Although a business process model is not the same as an activity model, the models have many similarities, which we previously mentioned in Section 2.1.2. Several other authors used the dataset of Friedrich et al. as a benchmark. Some comparison results can be found in the overview paper of Bellan et al.[19].

Furthermore, Bellan et al. investigate the area of approaches to extract processes from texts, where most of the approaches use NLP techniques to build business process models or similar models. In their discussion, they note that the dataset of Friedrich et al. is in several viewpoints quite limited, but at the moment of writing there are not datasets available with higher quality. The main criticism is that the dataset is not a representation of real-world requirements texts, the standard validation steps for a dataset are not executed and the correspondence between text and resulting business process model structures is also missing. Nevertheless, the dataset is a good first approach. Next to that, Bellan et al. state that a qualitative dataset might be too ambitious, but it would give the research field an enormous boost. A qualitative dataset would consist of process descriptions, business models and annotated texts. Another important statement is the fact that defining smaller transformation tasks will also improve the overall quality and research opportunities[19].

As a solution to the limitations stated by Bellan et al. concerning the datasets [19], they propose the PET dataset, which is an annotated dataset of existing requirements documents. The dataset consists of process descriptions, where the activities, gateways, actors and flow information are annotated. The data from Friedrich et al. has been used as a starting point[24], which is a well-known dataset in the community and it was not yet annotated. An important limitation of the diagrams of Friedrich et al. is that they were not validated by experts, therefore Bellan et al. do not include the diagrams in the PET dataset. The dataset contains 45 documents with a narrative description and annotated elements. The paper is at the moment of writing in pre-print, therefore we will not rely heavily on the paper, but we will use it as a reference work[33].

As we have shown there is only one available dataset with requirements documents and business process models, the dataset of Friedrich et al.[24]. The reuse of the dataset from Friedrich et al. in the PET dataset is a good contribution for the future and will make the evaluation process in the future more sound[33]. Unfortunately, the dataset is at the mo-

ment of writing not been reviewed by other researchers, which makes it at the moment potentially incomplete. The incompleteness might introduce changes in the future, in case of a change it would make our evaluation incorrect. Therefore we will make use of the dataset by Friedrich et al. and in the future might reevaluate our results with the PET dataset.

Chapter 3

Methods

In this chapter, we will define the different tasks we have encountered in the transformation process. For each task, we will propose an approach to execute the task, so that we can transform parts of the text into an activity model. All these different approaches combined make up our transformation pipeline. The pipeline can transform requirements text into activity models. In the first section, we will define the tasks and show how these tasks and in what order make up the pipeline.

3.1 Transformation tasks

As mentioned in the introduction we have developed a pipeline to extract information from the requirements texts, to build an activity model. The information that we are extracting can be divided into four tasks. With these different tasks, we can collect the information we need to build an activity model. We specify the following processing tasks:

1. Action extraction
2. Condition extraction
3. Reference resolving
4. Sequence identification

In the next sections, we will further elaborate on the different processing tasks and clarify their goal, the results it generates and how we have implemented them. All of these tasks combined make up our transformation pipeline. To give a clear overview of how these tasks interact with one

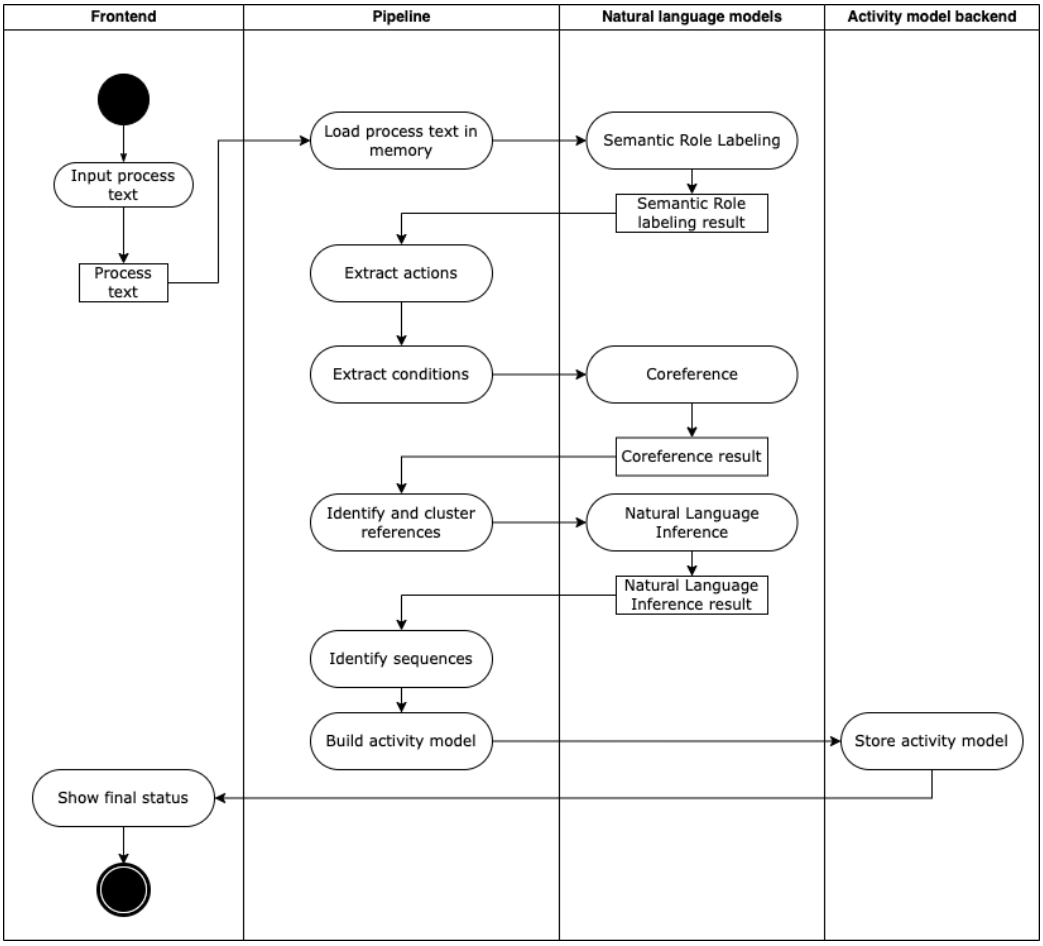

Figure 3.1: *Pipeline overview.*

another we have created an activity model, which is shown in Figure 3.1. Each action in the activity model contributes to one or more NLP tasks.

Each action in the model contributes to a processing task and each of the swimlanes depicts a system where the action is executed. We have four different systems. The front-end is the application that enables a user to interact with the pipeline. The pipeline system is the process that encapsulates all the transformation actions. The natural language models provide the NLP models that are used in the solution, such as semantic role labelling, coreference resolution and textual entailment. The activity model backend is the system where the final activity model is stored. In the following sections, we will dive deeper into what each processing task does and how it contributes to the whole pipeline.

3.2 Action extraction

Actions are executable atomic computations, which were briefly explained in Subsection 2.1.1. The action nodes can change the state of an activity model through an execution, thus it specifies the actions executed in an activity model. The action nodes are the building blocks of an activity model. For our goal to build an activity model from an unstructured text, we need to be able to extract these actions from a text and specify them in our activity model.

Several approaches in literature have earlier tried to extract actions from texts. Friedrich et al. extract actors, verbs and objects separately and in a later step combine them to create actions. They make use of the Stanford Parser, Verbnet and FrameNet to identify actions through grammatical relations[24]. Quishpi et al. extract the actions after the conditions have been selected. The actions are selected through a regular expression method called Tregex and extracted using semantic role labelling[35].

3.2.1 Extraction explanation

In our approach, we chose to use semantic role labelling (SRL) for the selection and extraction of action nodes. The choice for SRL enables us to handle more complex sentence structures and give a semantic meaning to that sentence. This information allows us to make extraction decisions for our actions and conditions. An explanation of what SRL is can be found in Section 2.2.1. Based on the roles that SRL can predict in a sentence, we can extract an actor, verb and object. The roles also give us more information than just an actor, verb and object combination, such as the adverbial. To

give a clear idea about the transformation process we show in Example (1) a sentence and in the sub-examples its' rolesets.

(1) A customer brings in a defective computer and the CRS checks the defect and hands out a repair cost calculation back.

a. [A customer] [brings] [in] [a defective computer] ...
ARG0 VERB ARG3 ARG1

b. ... [the CRS] [checks] [the defect] ...
ARG0 VERB ARG1

c. ... [the CRS] ... [hands] out [a cost repair calculation] .
ARG0 VERB ARG1

Example sentence[32] with its' SRL rolesets.

To transform these rolesets into actions we will select the actor, verb and object and everything in between those in the roleset. Example (1-a) will start with Arg0 and the end will be Arg1, such that it includes all found roles. For Example (1-b) it is a similar to the extraction process for the previous example. For Example (1-c) it is a bit more tricky, because there is overlap with Example (1-b). In Figure 3.2 we demonstrate the rolesets as action nodes based on Example (1).

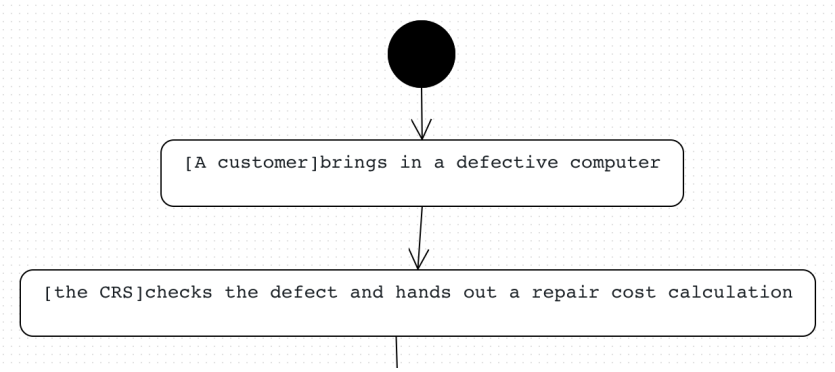

Figure 3.2: *Action nodes transformed from a sentence and its' rolesets.*

There are multiple approaches to extracting the information from a SRL result. One approach is to only select the words that correspond with roles and remove all other words. The limitation of this approach is that we will lose meaningful parts of the sentence that might be important to understand the resulting actions. A second approach is to select the roleset and take everything in between. The challenge of this approach is to define a way to choose between certain role sets and in the case of an overlap to combine or split them. We chose to use this second approach because

it gave the most meaningful actions and surrounding information about these actions.

To solve the problem of which roles we should select we have chosen to select all individual rolesets and in the case of overlapping rolesets, we combine them into a larger action. Example (1) has two of these overlapping actions, that are Example (1-b) and (1-c). We demonstrate in Figure 3.2 how these combined rolesets will show up in the final action. The data behind this combined action also holds the information of the found rolesets, such that we can use them in other transformation tasks. With this approach, we keep the most valuable information in the actions for later use. It might not be the correct action, but it does preserve important information that can be used by a human analyst to rewrite the actions. In the following subsection, we will describe how the transformation works on an algorithmic level.

3.2.2 Algorithmic extraction

First, we process the text using a SRL task, the results from the SRL are then used to extract the actions. In Algorithm 1 we describe how we select the different SRL results, that we will process into actions. The SRL results are grouped per sentence and within the sentence, there can be multiple role sets. The role sets can overlap with another role set, therefore we need to make a selection of the role set. The selection process is based on the roleset tags that are given in the SRL result.

Then with these identified rolesets we continue in Algorithm 2. Here we describe how we process all identified SRL results for a text and build actions. We have chosen not to include all functions that are referred to in the algorithm because these are not very important and the name gives an idea of what it does. Based on the identified roleset tags we identify several ranges and use these ranges to check if there is overlap between the rolesets. In the case, that we have found an overlap we make sure to grab the roleset with the longest tag length to make sure we include everything. There is one exception, in the case we have an adverbial, we will then split the result and tag them to reuse the information in the condition extraction process.

Finally, we end up with a selection of parts of the sentences and have information about the agent, verb, object and adverbials. These parts are in most cases the actions we end up with. The information is also processed by our reference processing and condition extraction. Where the reference processing mostly replaces some parts of the sentence with the

Algorithm 1 Extract agent,verb,object from SRL results

```
Require: list semantic_role_labelling_results (srl_results)
  procedure EXTRACT_AVO_FROM_TAGS(srl_results)
      text_result ← dict()
      for index, tag in srl_results do
          if tag ≠ "O" then
              if "begin_index" not in text_result then
                  text_result["begin_index"] ← index
              end if
              text_result["end_index"] ← index
              if "ARG0" in tag then
                  text_result["agent"] ← begin & end index
              end if
              if "B-V" in tag or "I-V" in tag then
                  text_result["verb"] ← begin & end index
              end if
              if "ARG1" in tag then
                  text_result["object"] ← begin & end index
              end if
              if "ARGM-ADV" in tag then
                  text_result["ADV"] ← begin & end index
              end if
          end if
      end for
      return text_result
  end procedure
```

Algorithm 2 Extract agent,verb,object from SRL results

```
Require: list semantic_role_labelling_results (srl_results)
  procedure EXTRACT_ACTIONS(srl_results)
      avo_results ← list
      for srl_result in srl_results do
          result_avo_results ← list
          for srl_verbs_result in srl_result["verbs"] do
              result ← extract_avo_from_tags(srl_verbs_result)
              if "ADV" in result then
                  result ← remove_conditional_keyword(result)
              end if
              append result to result_avo_results
          end for
          ranges ← combine_or_split_results(result_avo_results
          for range in ranges do
              sel_result ← select_result_range(range,result_avo_results)
              append sel_result to avo_results
          end for
      end for
      return avo_results
  end procedure
```

corresponding reference and the condition extraction mostly interprets the found condition and action. In the next section, we will further elaborate on the references and in Section 3.4 we will explain how we extract conditions.

After these processing steps have been finished we will transform the found sentence results into actions. This is a straightforward process because the processed text is added to an action node. The creation of the action node finalises the action extraction process. In Chapter 4 we demonstrate how our solution is implemented. For the SRL process, we made use of SRL BERT which is a transformer-based SRL model and was implemented by Shi et al.[12].

3.2.3 Limitations

The approach we have presented here is bound to some limitations. First of all the selection of the actions. For each sentence, we extract role sets and transform these into actions. The selection of these roles is hard because there can be information in there that we do not need. Therefore we have chosen to select all the information to make sure we do not lose any information, but this does not guarantee correct actions.

Another problem based on the identification is the splitting of actions. Where does a roleset specify the end of an action and should we continue with the next roleset. There might be actions that should be a combination of multiple rolesets and actions that should be smaller than a roleset. For this problem in a conditional structure, we have implemented an approach that combines separate role sets if there is a 'and' between them and it is part of a conditional structure. We further describe this in the Section 3.5.5.4. Still, this is just one approach, we have seen that there are several corner cases that need to be addressed.

Next to these identification problems, there is also the limitation of the construction of actions. We have chosen to keep the text mostly original and only replace some personal pronouns for clarity (Section 3.3.1). By keeping the original text we do not lose any textual information, but the actions can become long and incorrect. Other approaches have chosen to convert the sentence part to an active form, to make sure the actions stay concise. This transformation can cause some information to be lost. We have chosen to not do such a transformation, to keep most information available in the actions and enable the modeller to easily understand the actions. Still this is in some ways a limitation for the construction of activity models.

3.3 Reference resolution

In the process of building a model, the different parts of text are extracted from their context. An example of this is the creation of actions, we create an action by extracting it from the text. With that extraction, the surrounding text is not there anymore. Thus an action is removed from its context in the whole text. The extracted action can have several references to or within the context. These references were clear in the context, but without the context the references in the part of text become unclear or are lost in the process. Therefore it is important to identify where references refer to and make sure they are clear. To determine these references and their referred entities we make use of the NLP task coreference resolution. We have already explained the theory of coreference resolution in Section 2.2.2.

We identify the different entities using a coreference algorithm developed by Lee et al. [36]. The algorithm is implemented in the SpanBERT model, a transformer-based model, which was created by Joshi et al.[37]. We make use of the implementation library of AllenNLP, which we further describe in Section 4.3.1. The model can predict spans of text and couple them to entities they refer to[37]. The outcome of the model is a set of spans of text that refer to their antecedents in the text. Based on this outcome we can solve a few tasks:

1. Resolving personal pronouns to entities.
2. Couple similar references to one another.
3. Identify actors.

The approaches that we propose for each task are not completely standalone. Some of the tasks overlap with one another.

3.3.1 Resolving personal pronouns to entities

Personal pronouns refer to certain entities in a text. If these personal pronouns get extracted from their context it is hard to identify which entities they refer to. Therefore we need to be able to resolve the personal pronouns to the entities they refer to, which is called an antecedent.

(2) If the customer decides that the costs are acceptable, the process continues, otherwise she takes her computer home unrepaired.
Coreference example sentence[32].

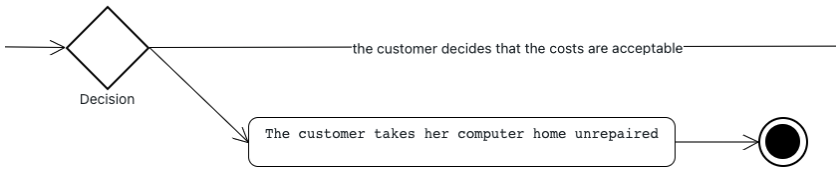

Figure 3.3: *coreference example*

Algorithm 3 Resolve personal pronouns to entities

Require: **dict** coreference_result, **list** avo_sents
  reference_results ← select_words_refer_antecedents(coreference_result)
  found_pers_pronouns ← select_personal_pronouns(reference_results)
  main_antecedents ← find_main_antecedents(found_pers_pronouns)
  tag_pronoun_antecedent(avo_sents,main_antecedents)
  for avo_sent in avo_sents **do**
    avo_sent[“node_text”] ← replace_text_coref_result(avo_sent)
  end for

To demonstrate this problem we show Example (2) with the references to the same entity underlined, namely “the customer”. In Figure 3.3 we show how this sentence with the personal pronouns replaced would look like in an activity model. In the action that models the part of the sentence “otherwise she takes her computer home unrepaired” we have replaced the word “she” with its antecedent “the customer”. It is possible to still understand the model with the word “she” because in the alternative path there is a mention of the word customer. In the case that there would be multiple alternative paths or with several actions in a larger process, it might be unclear to which entity “she” is referring. Therefore we have replaced it with the entity it refers to and will do this for all personal pronoun examples.

For the personal pronouns, we would like to know which entities they refer to. This makes it possible to replace the personal pronoun with the referred entity. The replacement will make it easier to read and understand the sentences and the resulting action nodes. There are several personal pronouns we can look for and replace, but not all personal pronouns can be replaced in such a way. Therefore we have specified a subset of personal pronouns we would like to replace, which are ‘i’, ‘we’, ‘he’, ‘she’, ‘you’, ‘they’, ‘it’. Based on these pronouns we replace them with their referred entity. In Algorithm 3 we describe our approach to replacing these personal pronouns.

We show in Algorithm 3 how we process the coreference result from AllenNLP to be able to replace the personal pronouns with their antecedent. First, we select all found references and filter them on personal pronouns. Then we select the antecedents of the personal pronouns, in the result, we keep a combination of antecedents and the pronouns. The antecedents we select are the "main" antecedents, with this we mean antecedents that are the main entity. As there are cases where a personal pronoun in one sentence refers to another. With the "main" antecedent we keep traversing the tree until we found the first mention of the antecedent. This does not ensure the antecedent is not a personal pronoun, but in most cases, it is replaced. Then we tag the pronoun-antecedent combination in the avo_sents. The avo_sents are all results that we have selected from our SRL result as possible actions. Finally based on the found pronoun-antecedent combination we build the text for the node based on the text with the reference replaced by its antecedent. In the case there was not a coreference in the avo_sent, we keep the original text. Through this approach, we can resolve the different personal pronouns and build understandable action nodes.

3.3.2 Cluster references

For condition extraction, we need to understand if certain sentences refer to the same entity. Therefore our approach needs to be able to cluster the different results that we use within our transformation process. To be able to do this we make use of the coreference results, we generated for the personal pronouns.

In Algorithm 4 we demonstrate our approach to tag the different clusters in our avo_sent results. We process the cluster data we receive from the AllenNLP coreference model. Then we order the coreference results per sentence, with the order we go through the avo_sents and check if there is an overlap with the range of the cluster and the avo_sent. If there is an overlap we append the cluster id to the avo_sent. Through this approach, we can tag the different clusters in our results. In the condition extraction, we further explain how we use the cluster id to link the results together.

3.3.3 Actor identification

To understand the responsibilities within an activity model it is important to identify the different actors in a model. With these responsibilities,

Algorithm 4 Cluster results

```
Require: dict coreference_result, list avo_sents, string text
  clusters ← coreference_result["clusters"]
  cluster_per_sent ← organise_cluster_per_sent(clusters,text)
  for avo_sent in avo_sents do
      if avo_sent["sent_index"] in cluster_per_sent then
          sent_range ← range(sent_begin,sent_end)
          for cluster in cluster_per_sent do
              if overlap cluster.range and sent_range then
                  append cluster.id to avo_sent["coref_id"]
              end if
          end for
      end if
  end for
```

we can create swimlanes, which we briefly discussed in Section 2.1.1. We have not implemented swimlanes in our model and editor but could do this with the actors that we have identified. From a sentence structure perspective, this is often the subject of a sentence. To be able to extract such an actor we can use the SRL results from our AllenNLP model. In Section 2.2.1 we demonstrated the general definitions of all roles in a SRL result. The tag Arg0 in the roleset defines the proto-agent role, that is in most cases the entity that executes the action defined in the role. Therefore as a way to identify an actor in an action, we make use of the Arg0 for each SRL result.

The identification of actors was not a goal of our solution, therefore the solution we present is an example and not a perfect solution. For good results, we suggest that Named Entity Recognition and Coreference clustering should be added. In our prototype, we do not make use of this, because our main focus is on building activity models.

In Algorithm 5 we demonstrate how we select the actors. It is a fairly straightforward process. Within each avo_result there is a collection of the outputs from the SRL tagging. We use these outputs to extract the data for the actors and use the Arg0 selected by the SRL tagging. Finally, in the process of building a node, we check if there is a "actor" key in the avo_sent. In the case that there is one, we remove it from the node text and add it in front of the node text and then build a node. This creates in a node a clear specification of the responsibility. This approach could be extended to build swimlanes, which are structures within an activity model to denote responsibilities.

Algorithm 5 Select swimlanes

```
Require: dict avo_results
  for avo_result in avo_results do
    for srl_result in avo_result["avo_results"] do
      if "agent" in srl_result then
        avo_res["actor_text"] ← srl_res["agent"]["words"]
        avo_res["actor_range"] ← srl_res["agent"]["range"]
      end if
    end for
  end for
```

3.3.4 Limitations

The limitations of the references are based on the three main tasks. For the resolving of personal pronouns, there are the incorrect personal pronouns that have been incorrectly replaced. In this aspect, some improvements should be on the specification of how we identify the personal pronouns and also on the model that predicts them. Another limitation with the personal pronouns is that in some cases the coreference cluster picks up an entity that should replace the personal pronoun, but the actor identification that we have done selects more than just the actor. Thus resulting in an incorrect action.

For the clusters, we found some examples where the incorrect cluster was predicted. On further investigation, we found out that it was not the fault of the model that predicted the cluster, but a human mistake in the specification of the antecedent. The pronoun was incorrectly specified as they instead of he or she, this problem is demonstrated in Section 5.3.2 at Problem 1.. Thus it is important to not forget the human aspect within these limitations.

Finally, our actor identification is a limitation, because some of the actors are incorrectly extracted. We already mentioned this previously, because it is just a demonstration. The incorrect actors are the result of our simple approach based on the Arg0 role from the SRL roleset. This needs to be further improved to identify the correct actors and make sure the actors are actors and not just the subject of a sentence.

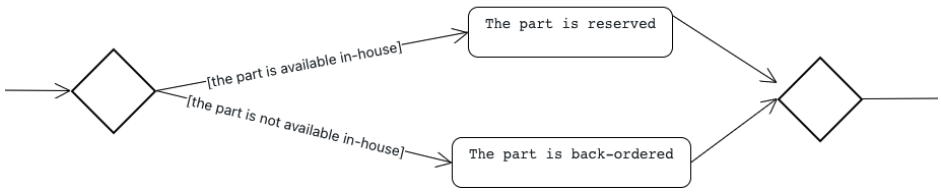

Figure 3.4: *Example of conditional nodes transformed from two sentences.*

3.4 Condition extraction

Within activity models, there are several types of nodes. One of them is a diamond-shaped node that specifies a decision node. The decision node is a node that can split the incoming flow into multiple flows. It has exactly one incoming flow and one or more outgoing flows. A merge node (similar shape) is the opposite, it can have multiple incoming flows and exactly one outgoing flow. On the edges (flows) between these nodes, there can be a text, which is called a guard. The guard depicts a condition that must be true to follow that path of the flow. In Figure 3.4 we show what such a decision node with its guards can look like.

In our transformation approach, we need to be able to identify a condition and its alternative conditions to model them in our activity model. With these conditions, we model them as a structure of conditions as shown in Figure 3.4. Each condition is a flow with a guard. Another important aspect is to identify which actions are part of a conditional flow and which belong to an alternative flow or should not be within the conditional structure.

In related literature, there are several approaches to extracting conditions and building certain conditional structures. In the preprint work of Hematialam et al., a model is proposed to predict conditions and their following actions in healthcare texts by using pre-trained transformer models and logistic regression[38]. Unfortunately, the models are specifically trained on medical prescription guidelines and would not perform very good on other types of text. Friedrich et al. make use of a rule-based approach, where they make use of conditional markers[24]. Their standard approach is when a marker is just a single marker, the word following it will be a gateway of the marker type e.g. condition. In the case of a conditional marker in a "mark" relation in the dependencies results from the Stanford parser, they consider the adverbial as a condition. Another ap-

Conditional markers	Words
Friedrich et al.[24]	if, whether, in case of, in the case of, in case, for the case, **whereas**, **otherwise**, optionally
Our markers	if, whether, in case of, in the case of, in case, for the case, optionally

Table 3.1: *Overview of conditional markers*

proach from Ferreira et al. defines several rules to identify certain business process elements[39]. The rules make also use of signal words and specific language structures to identify BPM gates, which are similar to decision structures.

A generic approach would be to build and train a machine learning model to predict conditions and their actions. Unfortunately, as we mentioned in Section 2.4 there are not many datasets of requirement texts available to train a model, therefore requirements texts with annotated conditions are even more uncommon. Based on the mentioned approaches and the lack of an annotated requirements dataset, we will use a combination of NLP models and conditional markers.

In Table 3.1 we show the set of conditional markers by Friedrich et al. and ours. Ferreira et al. define a similar set to the one of Friedrich et al. with some extra words, but we have experienced that the set of Friedrich et al. is sufficient. In the future, we might extend the set of words. We did remove some markers, which are 'whereas' and 'otherwise'. We still use 'otherwise' as a marker but in another way. We consider it to not have a condition following it because in most cases 'otherwise' defines an alternative action and does not introduce a new condition. In the next section, we demonstrate how we handle the 'otherwise' cases. Next to these indicators, we make use of the semantic roles provided by the SRL model from AllenNLP. Specifically, we look in these roles for adverbial clauses. We consider similar to Friedrich et al. that if we find an adverbial clause in combination with a conditional indicator we have found a condition and action. We also consider roles without the adverbial clauses, and we demonstrate this in the next section.

3.4.1 Condition identification

To demonstrate our identification process we will show two examples. Each with its' unique problems and context. After the two transformations, we will present an overview of our extraction algorithm in Section

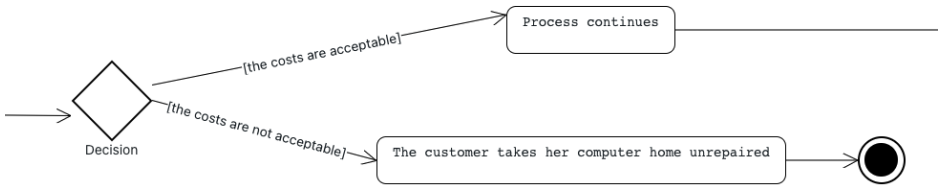

Figure 3.5: *Example (3) transformed into an activity model*

3.4.2. We first start with Example (3). The first line is the sentence that we will process then the sub-examples are the different SRL roles for the sentence.

(3) If the customer decides that the costs are acceptable, the process continues, otherwise she takes her computer home unrepaired.

a. If [the customer] [decides] [that the costs are acceptable] ...
   ARG0 VERB ARG1

b. ... [the costs] [are] [acceptable] ...
   ARG0 VERB ARG1

c. [If the customer decides that the costs are acceptable]
   ARGM-ADV
   [the process] [continues] ...
   ARG1 VERB

d. ... [otherwise] [she] [takes] [her computer] [home]
   ARGM-DIS ARG0 VERB ARG1 ARGM-DIR
   [unrepaired] .
   ARGM-PRD

Example sentence[32] with SRL rolesets.

In our transformation approach, we need to be able to interpret such a piece of text. As previously mentioned we make use of a list of conditional markers and semantic role labelling roles. The sentences in Example (3) can be transformed into an activity model that we show in Figure 3.5.

In Example (3) we can use the conditional indicator 'if' to identify that there is a condition. Then to extract the condition we make use of the first semantic role that follows the indicator which is Example (3-a). Then we need to consider Example (3-b), but it is a subset of the earlier example. So we skip it. The next role set Example (3-c) specifies an adverbial with the tag 'ARGM-ADV'. The adverbial specifies that there is a relation between the found condition in Example (3-a) and the next set of SRL tags in Example (3-c). This adverbial clause can be used to couple the condition and

following action. Through this approach, we can identify a condition and its following action. Then in Example (3-d) there is an alternative path for the condition. This alternative path specifies the condition "the costs are not acceptable", but does this implicitly. We identify this path, by looking for other conditional indicators, in this case, we have the word 'otherwise'. This indicates as we have earlier mentioned a conditional keyword that only specifies an action. After this sentence, we have found all the conditions and actions for the example. To define where we should end our search process, we search for flow terminations and alternative flows. We explain these concepts in Section 3.5.4 and 3.5.5.

Another example with two conditions in separate sentences is shown in Example (4) and (5). In Figure 3.4 we show the activity model that models these sentences.

(4) If the part is available in-house, it is reserved.

a. If [the part] [is] [available] [in - house] ...
ARG0 VERB ARG1 ARGM-LOC

b. ... it [is] reserved.
VERB

c. [If the part is available in - house], [it] is [reserved].
ARGM-ADV ARG1 VERB

Example sentence[32] with its' SRL rolesets.

(5) If it is not available, it is back-ordered.

a. If [it] [is] [not] [available], ...
ARG0 VERB ARGM-NEG ARG1

b. [If it is not available], [it] [is] [back - ordered].
ARGM-ADV ARG1 VERB ARG2

c. ... [it] is [back] - [ordered].
ARG1 ARGM-DIR VERB

Example sentence[32] with its' SRL rolesets.

Example (4) and (5) are two sentences that follow one another. The condition and action identification can be done in the same way as we did for the previous example. We use the conditional indicator, to identify the condition and the adverbial to find the following action. The difference in these examples is that there is not a clear definition of the alternative path. In the previous Example this was denoted by the word 'otherwise', but here the alternative path is the second condition. It is important to be able to identify the second condition such that we can use it as an alternative path. As a human, it is easy to see the contradiction between

the two sentences because the two conditions contradict one another. For a computer, it is not that easy as it needs some way to know the conditions contradict one another. This could be achieved through using the 'ARGM-NEG' tag which specifies a negotiation in the condition. Nevertheless, there can be situations where an implicit contradiction is made, in that case, there would not be an 'ARGM-NEG' available. To be able to handle these contradictions we make use of an NLP task called entailment or natural language inference. We have explained the theory of entailment in Section 2.2.3. With this model, we can compare the two sentences to identify if they contradict one another. In the case that there is a contradiction we have found an alternative path. In Section 3.5.5 we further explain how we use the result from the entailment task to identify these alternative paths and conditions.

The transformation examples we have demonstrated are the main approach that we use to transform the text into conditional structures for an activity model. Next, we will discuss how we do this in an algorithmic approach.

3.4.2 Algorithmic transformation

In the previous section, we have demonstrated how we can extract text and transform it into an activity model. In this section, we will demonstrate how we do this in an algorithmic approach. In Algorithm 6 we show how we identify the conditions and their following actions.

We first select the sentences with conditional keywords, which are the keywords shown in Table 3.1 with the empty conditional keywords added. In our case, this is only 'otherwise'. After we have selected the sentences and the begin and end index of the conditional indicators, we process this data per sentence and then per indicator. For each indicator, we search the SRL tags to try to identify an adverbial after or before the conditional indicator. In the case that we do find an adverbial sentence, we have the combination with the following action. We extract the begin and end index of the condition and action and add them to the results. For the case, where we did not find an adverbial we try to find a role set that is near the conditional keyword. In the case we found a role set, we denote this as the condition. The next part is to find the roleset that is just after the condition, in most cases, this is the action following the condition. There are cases where we can't find the following action then we leave the action empty and return the condition. In the case of an empty conditional keyword, we return an empty condition but do return a filled action.

Algorithm 6 Collect condition action data

```
Require: dict srl_results, string text
  condition_action_results ← list
  condition_sentence_data ← select_sentences_with_conditions(text)
  for sent_id in condition_sentence_data["sentence_ids"] do
      cond_ind ← condition_sentence_data["indicators"][sent_id]
      for cond_indicator in cond_ind do
          begin_index, end_index ← conditional_indicator
          adverb_data ← search_adverbial(begin_index,end_index)
          if adverb_data then
              action_data ← get_srl_result(adverb_data)
              condition_action_results.append([adverb_data, action_data])
          else
              cond_result ← get_srl_result(begin_index,sent_id)
              action_result ← get_srl_result(cond_result.end_index,sent_id)
              condition_action_results.append([cond_result, action_result])
          end if
      end for
  end for
  return condition_action_results
```

The action data we generated from the text and have shown in Section 3.2 is used in the condition and action process. We use the identified conditions and actions to tag them in the action data. An action data result is tagged if there is an overlap of the index between a condition or action that we have identified. We tag this result with the word 'condition' or 'action' accordingly. Furthermore, we also add the conditional indicator, to make sure we can reuse it if needed. In Section 3.5.5 we further specify how we handle multiple conditions and how we couple the correct ones using entailment. In Section 3.5.4 we demonstrate how we specify terminations in conditional structures.

3.4.3 Limitations

There are some limitations to the demonstrated condition extraction approach. First, there are examples where the condition or action is not correctly tagged. The approach is very dependent on the sentence structures and conditional keywords, but the dependency does not by definition provide the correct results. There could be other conditions and actions in one or more sentences. These limitations need to be taken into account when looking into the final results.

3.5 Sequence identification

Sequence or path identification is a complex task in the transformation of text into activity models. By sequence identification, we mean the sequence of actions that are specified in a text. Based on the possible structures in activity models and earlier work we define the following paths:

1. Sequential paths.
2. Repetition paths.
3. Parallel paths.
4. Termination paths.
5. Conditional paths.

We will describe for each path type how we identify and handle the paths. For the repetition and parallel paths, we have not implemented a solution, therefore we will describe a possible implementation. The conditional path is the most extensive because there we have put most of the effort into.

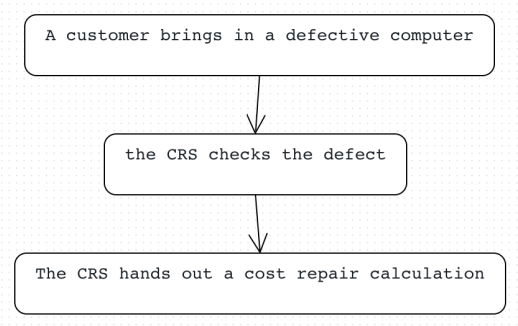

Figure 3.6: *Action nodes transformed from a sentence and its' rolesets.*

3.5.1 Sequential paths

By sequential path, we mean a path of actions that follow one another. One could argue these are the normal path cases, where it is clear which action follows the previous action and the order is chronological. In our case and similar to other research contributions, we assume that the text is written in chronological order. From this assumption we can process the actions in the order they are written when we go through the text from the beginning to the end. There are corner cases to this approach, for example with references that refer to previously defined actions. For all actions where we cannot identify another type of path, like conditional, repetition or parallel, we process them in sequential order. For our solution, this means that, when in a text first action 'X' is described and then action 'Y' is described, in the final solution action 'Y' will come after action 'X'. With this approach, we can make sure the chronological order of the text keeps existing and the sequential paths. To give an example of how this would look like we present Example (6) that we transform into the activity model shown in Figure 3.6.

(6) A customer brings in a defective computer and the CRS checks the defect and hands out a repair cost calculation back.
Example sentence[32] with its' SRL rolesets.

3.5.1.1 Limitations

The processing approach of sequential paths is fairly simple, as we take the actions in the order we have found them. A limitation to this approach are the actions that are processed through this approach but should have been processed in another way. Such as a repetition, condition or parallel

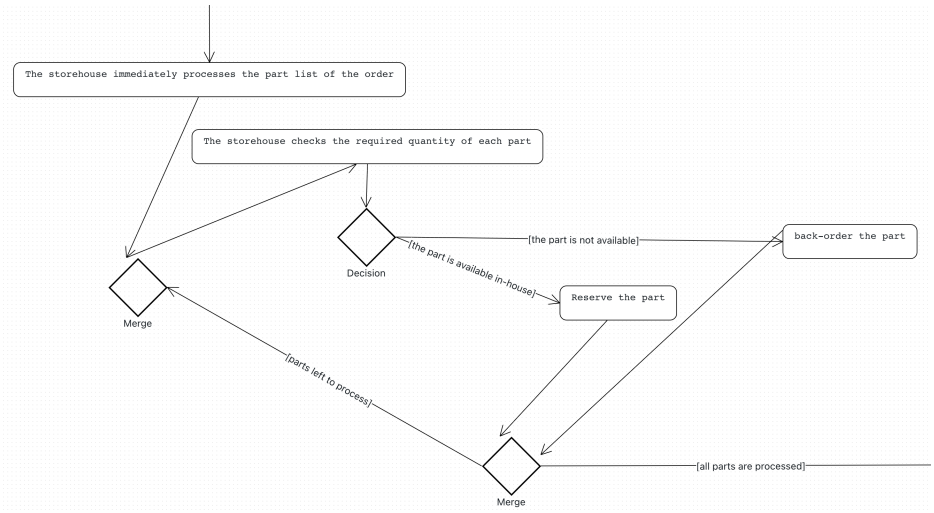

Figure 3.7: *An example of a repetition based on Example (7).*

way. This is the effect of our solution not being able to identify the correct paths in the text.

3.5.2 Repetition paths

Another important type of path in a text is repetition. A repetition refers back to an earlier mentioned action and makes sure the action or set of actions is repeated. Repetitions are also present in activity models, therefore it is important to be able to identify these. In our approach, we did not implement this approach but defined a way it could be implemented.

(7) The storehouse immediately processes the part list of the order and checks the required quantity of each part. If the part is available in-house, it is reserved. If it is not available, it is back-ordered. This procedure is repeated for each item on the part list.
Example[32] sentences for a repetition.

To process a repetition, as shown in Example (7), it is important to first identify that something has to be repeated. Then we need to understand what will be repeated. So the problem is two-fold, an identification problem and a reference problem. To demonstrate how this would look like in an activity model we present Figure 3.7. Here we have presented the repetition path with a guard returning to the earlier decision node where the parts are being reserved and ordered.

For the identification problem, we can make use of keywords that spec-

ify a repetition. In Friedrich et al. they specify the following set of repetition words: 'next', 'again' and 'back' [32]. Besides these keywords, we propose to add the word 'repeat'. With these words, we can identify that there is repetition within a certain sentence. For example in Example (7) this is the verb 'repeated'.

Then the next part is to resolve the reference to the action that should be repeated. In Example (7) this would be the process of checking the required quantity of each part and then reserving or back order it. The reference to the earlier action is indicated by the word 'this'. We need to be able to couple the word 'this' to the earlier action. The most common approach to identify this is to use a coreference model to resolve the reference. Such reference resolutions are good at processing references to persons and entities. An important aspect of repetition references are demonstrative pronouns, which are pronouns such as 'this', 'that', 'these' and 'those'. However as recent research has shown the performance to resolve demonstrative pronouns is quite poorly, which is mostly caused by the low number of demonstrative pronouns in the training datasets in comparison to other pronouns[40]. Many repetition references are denoted using demonstrative pronouns, therefore the low performance makes this approach not usable. Another approach could be through using certain relations between the sentences, but this has to be on the text level instead of the more common sentence level. We could use only the references that coreference is currently able to handle, but then a large number of repetitions will be missed.

To be able to process repetition paths with certain confidence we need better tools and models to annotate and understand a text. In the case that coreference is also able to handle demonstrative pronouns, the approach can be used to handle repetition with more confidence than we currently can.

3.5.3 Parallel paths

Parallel paths are two or more paths (flows) that are executed simultaneously. To identify the path that specifies the parallelism we can also make use of keywords. The sentence that specifies the parallelism is in most cases referring back to previous sentences. The challenge is to identify the sentences and actions it refers to, to be able to create a correct parallel structure. To show this challenge we present Example (8).

(8) Hotel example with a parallel path.

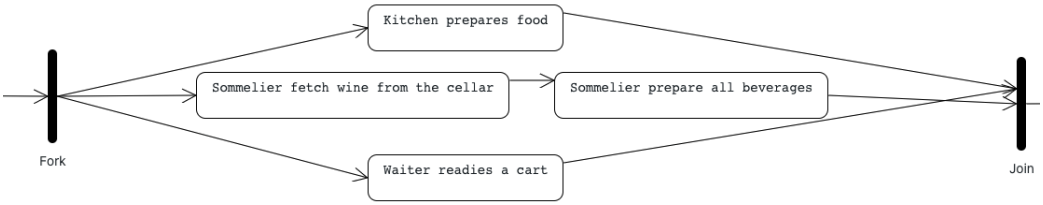

Figure 3.8: *Example of a parallel structure.*

a. She then submits an order ticket to the kitchen to begin preparing the food.
b. She also gives an order to the sommelier (i.e., the wine waiter) to fetch wine from the cellar and to prepare any other alcoholic beverages.
c. Finally, she assigns the order to the waiter.
d. While the kitchen and the sommelier are doing their tasks, the waiter readies a cart (i.e., puts a tablecloth on the cart and gathers silverware).

Example[32] sentences defining a parallel path.

In Example (8) multiple sentences are part of the repetition. The sentences are stated in this order, one sentence has been removed because it was a sentence that did not describe an action. In sentence (8-d) the parallelism is specified and the reference to the earlier sentences are constructed, with 'the kitchen' and 'the sommelier' referring the to same entities in sentence (8-a) and (8-b), therefore the tasks mentioned in sentence (8-d) are the tasks performed in sentence (8-a) and (8-b). Next to the specification in sentence (8-d), there is also an action performed by the waiter, this action is being performed in parallel to the referred actions. If the sentences were transformed into an activity model, this would look like the activity model shown in Figure 3.8.

Our approach to process this would be to identify the sentence with a parallel keyword: 'while'. Then we extract the sentence structure through SRL results and use a coreference model to identify the three entities: 'the kitchen', 'the sommelier' and 'the waiter'. The SRL results enable us to identify the parallel actions, that can be found in the SRL role that follows the parallel keyword. Then the SRL enables us to identify the second action in the same sentence. To combine all referred sentences we can use a coreference model to find the previous references to the entity mentioned in the parallel sentence shown in sentence (8-d). A problem with this approach is where should we stop identifying sentences that should be part

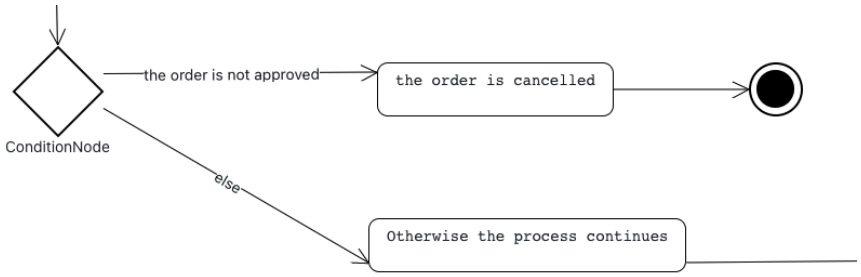

Figure 3.9: *Example of a condition with a termination.*

of the parallel system. In the text where Example (8) is taken from the sentence before sentence (8-a) does not contain a reference to any of the entities that we consider in sentence (8-d). With this knowledge, it would be easy to state that we can stop, if we do not find any references to the entities we are considering. Although at first, this might seem like a good solution it is not a generic one, because there could be certain references that should not be on the parallel path. Therefore the proposed approach would not transform the text into a 'correct' solution and might even introduce more vagueness to the resulting activity model.

A way to solve this would be to understand the text and understand where the parallelism starts and stops. There might be an approach that we have not considered yet, but they do not exist in our current understanding. Future improvements with transformer models might be able to provide better solutions. Also, the current developments in the datasets in the research area, as mentioned in Section 2.4, are a promising area to provide a result for this solution.

3.5.4 Path terminations

Path terminations are places where the flow stops. These are important to specify because there are places where the process should no longer continue. In Figure 3.9 we show an example of what such a termination would look like.

The black circle in Figure 3.9 is the node that ends the activity flow. The termination is clearly stated with the word 'cancelled', as it shows the process needs to stop. This can easily be handled with a keyword-based approach. Unfortunately with a keyword-based approach, we are not always sure to terminate a process, because there can be scenarios where the termination keyword does not mean to terminate the flow. For example in

the sentence in Example (9).

(9) If the storehouse has successfully reserved or back-ordered every item of the part list and the preparation activity has finished, the engineering department assembles the bicycle.
Example[32] sentence with a non termination keyword.

In the shown example the word 'finished' does finish the preparation activity, but the process is not finished. Another example where we should identify a termination is in the case of an implicit end of the process. In Example (10) we show a sentence with an implicit termination.

(10) If the customer decides that the costs are acceptable, the process continues, otherwise she takes her computer home unrepaired.
Example[32] sentence with an implicit termination.

The part following the word 'otherwise' introduces an implicit termination. The implicit termination is clear for a human to understand, because of the context of the sentence. For our system, there is not a clear end to this process. This is important to realise because we would need to understand the context to identify this termination. For both Example (9) and (10) we need further context to understand if we should stop the process.

For our implementation, we are not going to solve the problem for Example (10) because we need a clear context to create a termination and we do not have the models to do this. For Example (9) we can process it by skipping this mention of 'finished'. To do this we have chosen to only process termination keywords if we are in a conditional structure that has at least a second flow to continue the process and the termination happens inside an action. In all other cases, we skip the termination keyword. This way we can handle terminations in the clearest examples and do not terminate the process too early.

To identify the final node we make use of a list of termination keywords. The list might need to be updated in the future, but for now, we consider the following words: 'cancel', 'finish', 'stop'. Using these keywords we can identify terminations and process them accordingly.

3.5.4.1 Limitations

The first limitation is the choice to only create termination nodes when we have a conditional path. There should also be terminations in the model when we are dealing with a normal path. We chose to not do this because the model would create loose ends. Thus the model would not be

completely linked and in some cases, this must be. So here we need to improve our approach, to make sure we can also create terminations in other cases.

A second limitation is the way we identify the terminations. We chose to use keywords, and stick to paths that have an alternative path. To make this approach more robust, there should be a way to identify the meaning and understand if a termination keyword really terminates the process. Through that approach, we can create better terminations within the model.

3.5.5 Conditional paths

A conditional path is a flow with a guard and some actions, the flow starts at a decision node. Conditional paths are a set of multiple of these flows that offer multiple views on the same subject in a condition. In Section 3.4 we already discussed our condition identification and extraction approach. Here we will further explain how we combine the identified conditions and make sure they are conditions that refer to the same subject. We have identified three types of conditions in a text that we need to be able to handle. These different types each have their approach to compare them and identify if they need to be merged using a decision node.

1. Sequential conditions.
2. Conditions spread throughout the text.
3. Conditions without an explicit alternative condition.
4. Actions in a conditional path.

For each of these types, we will demonstrate an example in text and then how we combine the identified conditions using our coreference and entailment approaches.

3.5.5.1 Sequential conditions

With sequential conditions, we mean conditions that follow one another in the text. To demonstrate such a condition we show Example (11). Where the first condition is stated in the first sentence and in the next sentence the second condition is stated.

(11) Conditional sentences from the bicycle example [32].

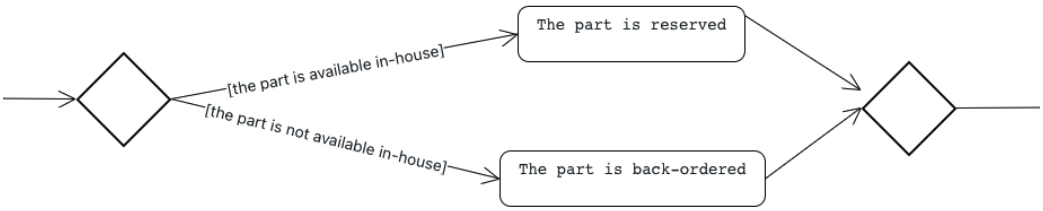

Figure 3.10: *Example of conditional nodes transformed from Example (11)*

a. If the part is available in-house, it is reserved.
b. If it is not available, it is back-ordered.

Example[32] sentences to show a sequential conditional structure.

The sentences from Example (11) can be transformed into an activity model, which we show in Figure 3.10. In Example (11) we have underlined the two conditions. As previously mentioned with our approach we can identify these conditions and their actions. The second sentence in the example directly follows the previous sentence in the text it is taken from. Therefore we can with almost certainty identify that they are referring to the same condition and defining two conditional paths. To ensure the sentences contradict one another, we make use of textual entailment, which we explained in Section 2.2.3. We compare both conditions with the textual entailment model, which can predict if the sentences are a 'contradiction', 'entailment' or 'neutral'. We store the prediction from the textual entailment in the data we keep about the conditions. For the case, we found a contradiction we combine the two conditions with a decision node. The actions we found next to the condition we push behind the decision node and its corresponding condition guard.

3.5.5.2 Conditions spread throughout the text

Another type of condition is a set of conditions where the condition sentences have other process text in between them. For example when condition 'A' has been stated, then a lot of other process text is described for that particular process, after that condition 'B' is defined. In that case, we need to be able to combine conditions 'A' and 'B', to build a correct conditional structure. To give a clear idea we use Example (12).

(12) Conditional sentences from our example.

a. A customer enters an order.
b. If the order total is more than 10.000 euros, the order needs to be approved by the manager.
c. If the order is not approved, it is cancelled.
d. If the order is approved, or the total is less than 10.000, the inventory manager allocates the stock.
e. If the stock level is too low, the product is reordered.

Examples of text with conditions.

Within the example, we have underlined the two conditions we mean to show as an example. The conditions are not directly following one another, because there is another sentence in between and even a second condition structure, which is the order approval and the scenarios where it is approved or not approved. The order approval condition can be processed using the technique we previously proposed for Example (11). The condition that we have underlined needs to be solved differently. To do this we need to understand that the conditions are focusing on the same subject, in this case, 'the order total'. We can identify this reference using a coreference model. From the coreference model, we can use the found coreference clusters which we discussed in Section 3.3.2. The coreference clusters are tagged in each action result, therefore we can use the action results that are tagged as conditions and if there is a cluster similarity we can compare the conditions. The comparison can be made using the textual entailment model. In the case of a contradiction, we can almost accurately combine the conditions. For such cases, we first try to find a follow-up condition, if we can't find any conditions in the sentence after its' specification we use the coreference clusters.

In Figure 3.11 we demonstrate how Example (12) would look like as an activity model. We have drawn a red box around the condition that we coupled to the earlier condition using coreference. With the entailment and coreference approach, we can handle more complex structures to build our activity models.

3.5.5.3 Conditions without an explicit alternative condition

Lastly, we need to be able to handle conditional paths if we could not find an alternative conditional path. In that case, we probably missed an implicit conditional as we also describe in Section 3.4.1. We know that our solution will not be able to give 100% correct solutions and as earlier mentioned this is not the goal. Therefore in the case, we could only find one condition we create an alternative flow to the merge node that

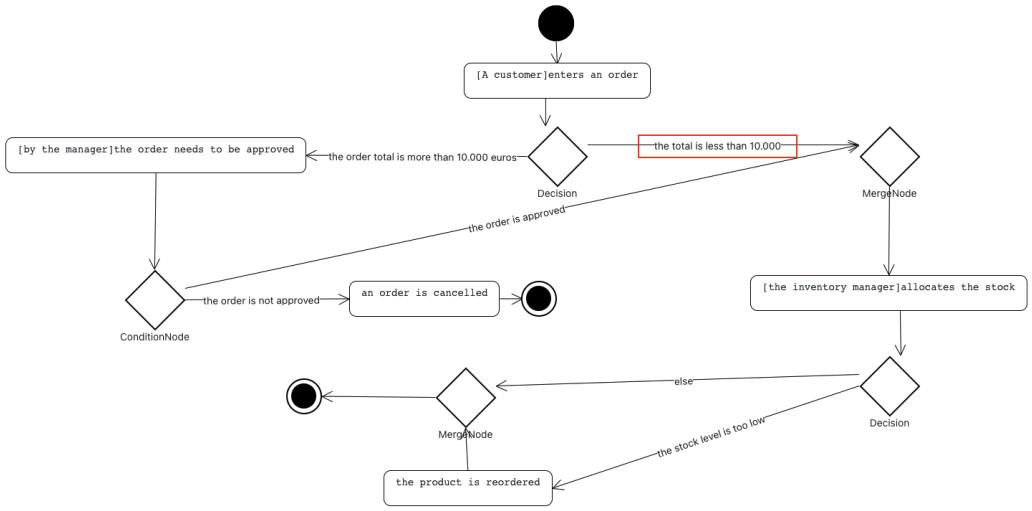

Figure 3.11: *Example of a condition with a termination.*

comes after the last action in the conditional path. This alternative flow will then be defaulted to have the guard '[else]' on it, with this flow our model will specify conditional structures and handle missing conditions. To view what this would look like we show an example in Figure 3.12, where we could only find one condition. The user can change the flow to point to the correct following action or change the guard in our editor, which we further demonstrate in Section 4.3.4.

3.5.5.4 Actions in a conditional path

In the previous sections, we have demonstrated how to combine the different conditions. Another important aspect of these conditions is to be able to identify where a conditional path stops and continues with the next one. In the previous examples, the end of a conditional path and action was clear, because we only considered the first found agent verb object result as the following action. In the scenario, where we found another conditional keyword we would consider it as an alternative conditional path. For the final conditional phrase, we need to be able to identify when we do not consider new actions. It is hard to identify this because there are no clear markers and easy ways to do this. Therefore we have chosen to only use the first found result. Through the tests that we ran, we found corner cases of different actions that should be part of the structure. Most of these results were coupled using conjunctions. In Example (13) we show such a conditional path with actions that should be combined

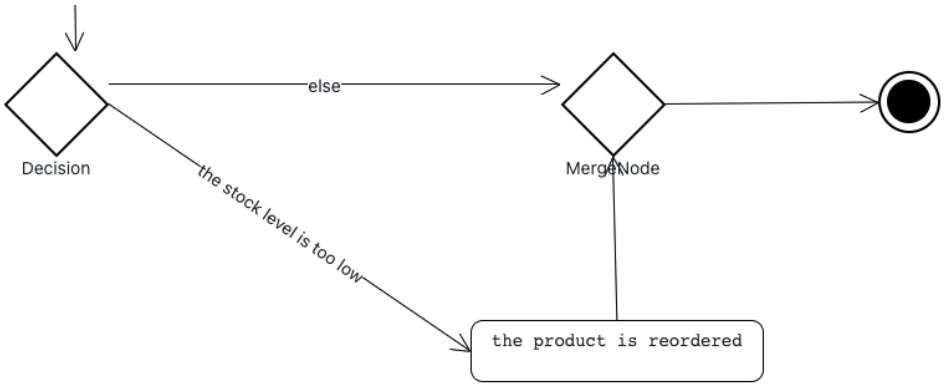

Figure 3.12: *Example of a condition with a termination.*

through their conjunction.

(13) a. If the assessment is positive, a garage is phoned to authorise the repairs and the payment is scheduled (in this order).
b. Otherwise, the claim is rejected.
Conditional example with a conjunction[32]

To be able to handle such cases we extended our approach with the search for a conjunction before stopping the search for a follow-up action. We do this by a comparison of each agent verb object result with the result that entails them. If there is an 'and' in between the two results in the sentence. We tag both results with a key to identify the relation and the conjunction word. The key specifies the beginning for the first action and the end for the following action with the words "and-conj-begin" and "and-conj-end". Through this approach we look for "and-conj-end" when building the conditional structure, if there is such a key in the agent verb object result we know it was coupled to the previous condition or action. With this approach we can build conditional structures such as in Figure 3.13, thus processing more complex conditional structures.

3.5.5.5 Limitations

The demonstrated approaches are quite good at comparing the examples and finding contradictions. A limitation of these approaches is that not all conditional structures always have contradictions. There can also be other statements in the conditions. Therefore the approach is in some cases, not the best. For the best results, an analysis of the whole text that returns the

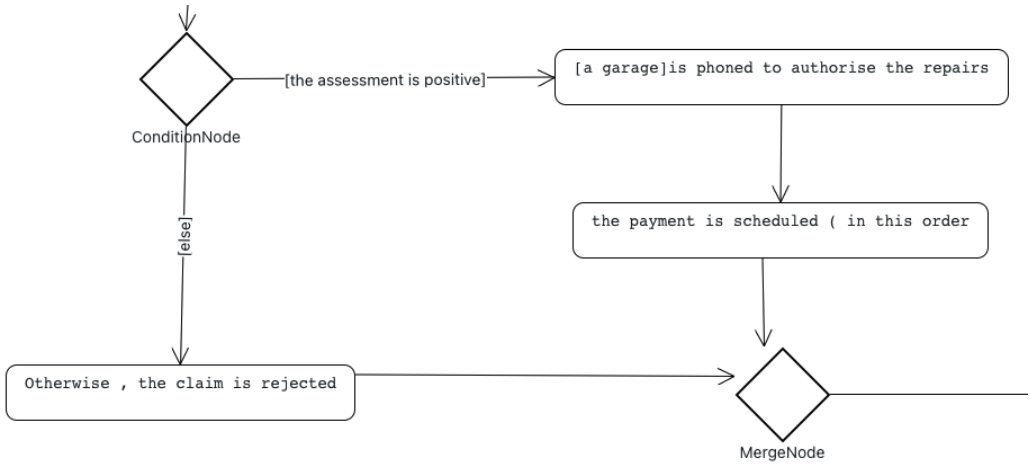

Figure 3.13: *Example of a conditional path with a conjunction between two actions based on example (13).*

paths within a text would be a better result for identifying the conditional paths and building the structures.

3.6 Activity model construction

After all of these transformation tasks, the final task is to combine all of the collected information into an activity model. The information is stored in a list called avo_sents, which we have mentioned in some of the previous sections. The data stored in the avo_sents is in chronological order. Therefore we go through the list of elements in avo_sents and create a node or multiple nodes for each element. The elements in avo_sents are dictionaries, such that we can store different types of information there.

There are several pieces of information stored in each avo_sents element. The most important ones for the creation of the model are:

1. Text for the node.
2. A tag for condition.
3. The conditional keyword.
4. A tag for the action following a condition.
5. The coreference cluster id.
6. The NLI value with its' neighbour.

7. The NLI value for the coreference neighbours.

8. The SRL results for all the data in this element.

We process each of the avo_sents elements in the order they are presented, which is the chronological order. In the case of a normal action, we process it by creating a new node and binding it with an edge to the previous node. Finally, we update the previous node.

For the cases that we have a 'conditional' key in the avo_sents element, we process a subset of the avo_sents starting at the element with a 'conditional' key. We use the information we presented in the enumeration for building everything. We keep track of all the created decision and merge nodes, such that we can use them for special conditions. After the conditional structure has been created we continue with the last element. The last created merge node is the node we set as the previous node.

In the case of a special condition that refers back to an earlier condition through reference, we use the set of created decision and merge nodes. We use it to connect our coreference condition back to the previous condition. Then we take the corresponding merge node and set it as the previous node. We then continue with the avo_sents result that follows the result we just used.

Finally, we go through the model as a whole and add to all the decision nodes with only one outgoing edge an extra '[else]' edge that connects to the first found merge node. After that we search the whole model for merge nodes with only one incoming edge, we remove these merge nodes and connect the incoming edge to the outgoing edge.

This process enables us to build an activity model based on the data that we have collected during our earlier described methods. Finally, this data is saved to the database in our system, which we will further explain in the next chapter on our system design.

Chapter 4

System design

In this chapter, we will specify how our system has been set up and which components contribute to the process within the system. Different from standard research practices, the software that we build will be used and form the basis for other research projects in the future. Therefore we had to build a system that can be extended, upgraded and reused. These requirements are the reason for the extensive description of the system design in this chapter.

To begin, we will explain the environment our system operates in, in Section 4.1, because our system is part of a larger software solution. Then we discuss the logical design of the solution in Section 4.2. Finally, we end the chapter with the technical design of the solution in Section 4.3.

4.1 Environment of the solution

The proposed solution is part of a larger environment. This environment is a combination of several projects, each to help users create better requirements. The whole environment is called P2P which is short for Prose to Prototype, which points to the written documents we transform into UML models. The vision for the P2P project is described in the paper of Ramackers et al., which describes the different components in more detail[41]. From the paper, we have taken the logical architecture for all the components and show it in Figure 4.1.

As shown in Figure 4.1 several components together create the whole P2P System. The solution we propose in this research makes use of many of these components and is an extension of some components. Specifically, our solution introduces new additions for the specification mapping, the

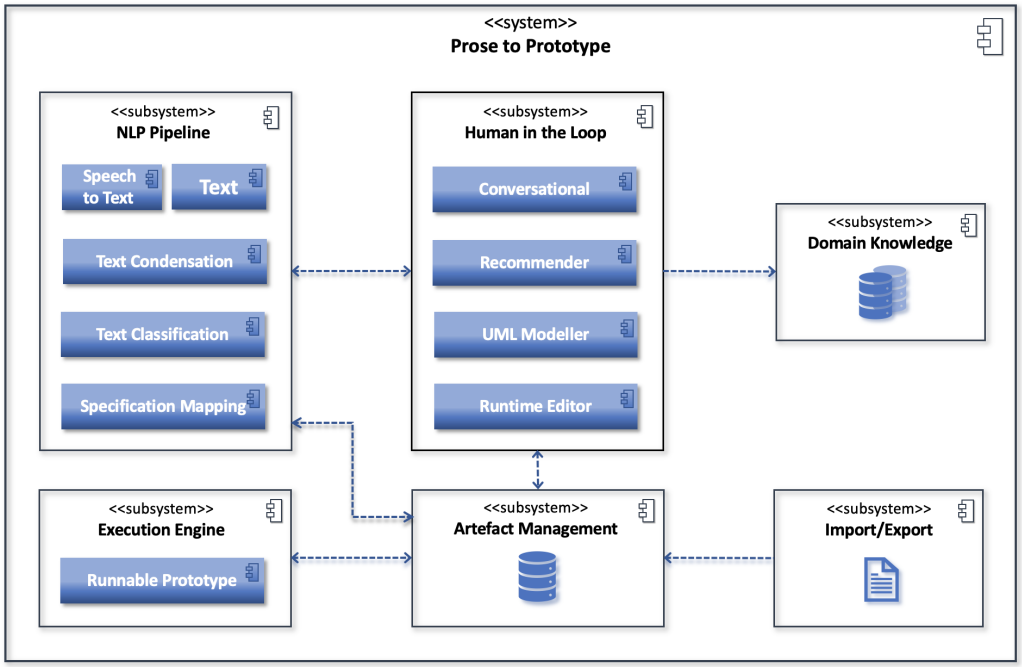

Figure 4.1: *Prose to Prototype Architecture[41]*

artefact management and UML modeller components. In the next sections, we will elaborate on how our solution is defined and adds to these existing components.

4.2 Logical Design

The logical design of our solution is an overview of the functions that our system provides. A schematic overview of this is shown in Figure 4.2.

The architecture consists of 3-tiers a presentation, application and persistence tier. Each tier has its' own responsibilities. In the following section, we will describe what each tier does and how it is part of the architecture.

4.2.1 Persistence tier

The persistence tier holds all the data for the applications. There are two main functionalities for the persistence tier. The first is to store all the requirements texts that have been given as input. These texts can be reused in other tasks and might be useful for comparison with the generated mod-

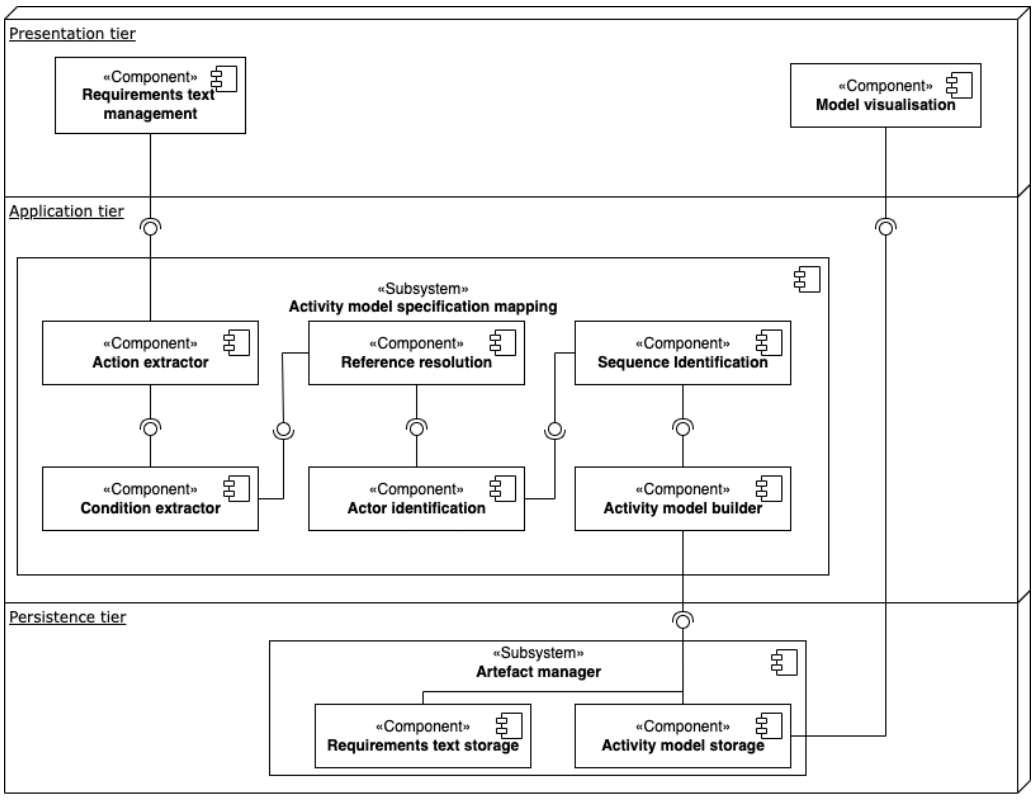

Figure 4.2: *Logical architecture of our solution.*

els. The second functionality is the storage of the generated activity models.

4.2.2 Application tier

The application tier is the tier where most of the processing takes place. There are several tasks performed in this layer. The first task is to extract actions from the text. Then we will extract the conditions and their corresponding actions, and tag them in the data we keep. After that, we process the different references and tag the found references in the data. Then we use the references to identify the actors in the text. With the found information we identify the different sequences in the text that are used to build the activity model flows. Finally with all the information from the previous steps we construct an activity model. The model is then stored in the persistence layer.

4.2.3 Presentation tier

The presentation tier has two tasks. The first task is the collection of the requirements text, which is used to start the processing of the text. The second task is the visualisation of the model, this will enable the user to see the extracted model from the text.

4.3 Technical Design

With the logical design from Section 4.2, we have proposed the different functionalities that our solution will fulfil. In the following sections, we will outline how we implemented these functionalities from a technical perspective. Our architecture is a combination of several technical components, that together make the transformation process of requirements documents into activity models possible. In Figure 4.3 we show what components exist and how they are dependent on one another. In the next subsections we will describe the subsystems, such as the Django Framework, AllenNLP and the P2P database.

The technical architecture in Figure 4.3 shows two main subsystems, the Django Framework and the AllenNLP system. Besides these two important systems, the P2P database and UML activity model editor are also important aspects of our solution. We run all of these components in Docker containers to make sure they run anywhere. This also allows us

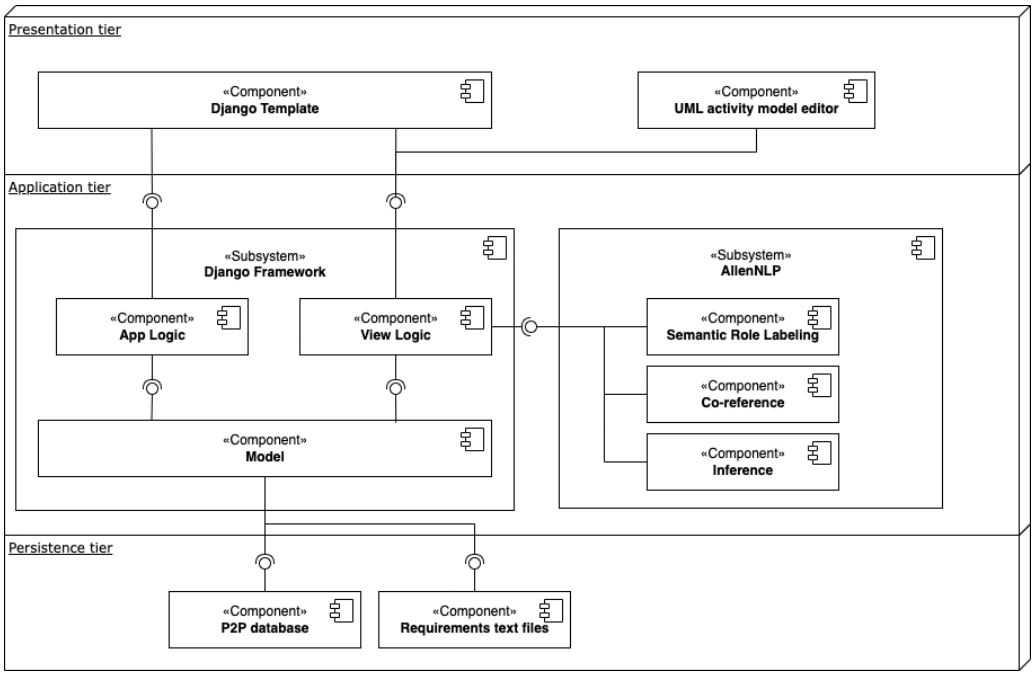

Figure 4.3: *Technical architecture of our system.*

in the future to deploy everything to external services to provide better resources for the resource-intensive services, like AllenNLP. In the next sections, we will go further into each of the systems that we have mentioned as important. We start with the NLP models from AllenNLP.

4.3.1 AllenNLP

To be able to run our system we make use of several Natural Language Processing models. There are several models and platforms available all with their specific benefits. A complex problem is to get each of the models working correctly, for the models we have chosen AllenNLP because they provide close to the state-of-the-art performance and are very reliable. To further elaborate, AllenNLP is an NLP platform that enables researchers to easily implement and use concise NLP algorithms in their research[42]. It takes the need to debug and work on the complex implementation problems away. In their platform, they provide several models for all kinds of NLP tasks. The tasks we use in our solution are mentioned and described in Section 2.2. For now, we make use of the semantic role labeling model from Shi et al. [12]. The coreference model is from Lee et al. [36] and the textual entailment model is built by Liu et al. [43]. These models can be

replaced by downloading new ones and updating the paths that load the models.

For our approach, we run the AllenNLP models in a docker container. The implementation with Docker enables us to move the models to almost any device. Each of the models has its own endpoint and is loaded using the AllenNLP libraries. There is an out-of-the-box AllenNLP docker image* available, but we have chosen to build the docker image by ourselves. This enables us to only download the preferred models and keep the whole container smaller than the out-of-the-box version which retrieves more models. We might switch to the out-of-the-box version in the future if the P2P solution will be using more AllenNLP models in the future.

4.3.2 P2P backend

The P2P backend is the system that is used for the communication between all the components. The framework was first introduced with the Django framework in the work of Driessen et al.[44]. In our research, the system is extended to be able to store activity models. The whole system can be almost directly mapped on the P2P Architecture in Figure 4.1, most of the systems shown in that Architecture are part of the system.

The work of Driessen et al. introduced the system, to generate and edit run-time applications from class models[44]. Tang et al. extended the system to incorporate an NLP pipeline to transform requirements texts into class models[45]. With this work combined the system was now able to generate an application based on a piece of requirements text.

To further extend this work for activity models we have implemented an activity metamodel, and API endpoints to create, read, update and delete the activity models and the activity model generation pipeline. The metamodel is defined in Section 4.3.3 where we show the different classes we have taken from the Object Management Group (OMG) Activity model standard. The different API endpoints can handle the data that is specified using the activity metamodel. The activity model generation pipeline is a combination of all demonstrated transformation approaches and the NLP models. Next to these contributions we also save all the requirements text to the backend, for future reuse and research.

*`https://docs.allennlp.org/main/#installing-using-docker`

4.3.3 Metamodel

As a target for our pipeline, we have defined a metamodel, the model allows our solution to save and retrieve the created activity models. The metamodel is a derivation of the UML standard for the activity model because the standard is too extensive for our solution. We have identified a set of basic classes which are most often used in activity models. We will first define the specification of our metamodel, then we will discuss some of the classes that are important to understand the metamodel.

4.3.3.1 Metamodel specification

In our implementation, we have chosen to select several classes from the activity model definition as it allows us to keep the model comprehensible and possible to work with. An overview of all the important classes can be found in Figure 4.4, we do not implement all of these classes, but demonstrate them to show their relations to other classes.

Based on the subset presented in Figure 4.4 we have implemented a subset of these classes, which are all the classes except for a few. The classes we did not implement are: Classifier, ActivityPartition, Behavior, ObjectNode, ActivityParameterNode and State. It is possible to add the excluded classes in the future for further implementation.

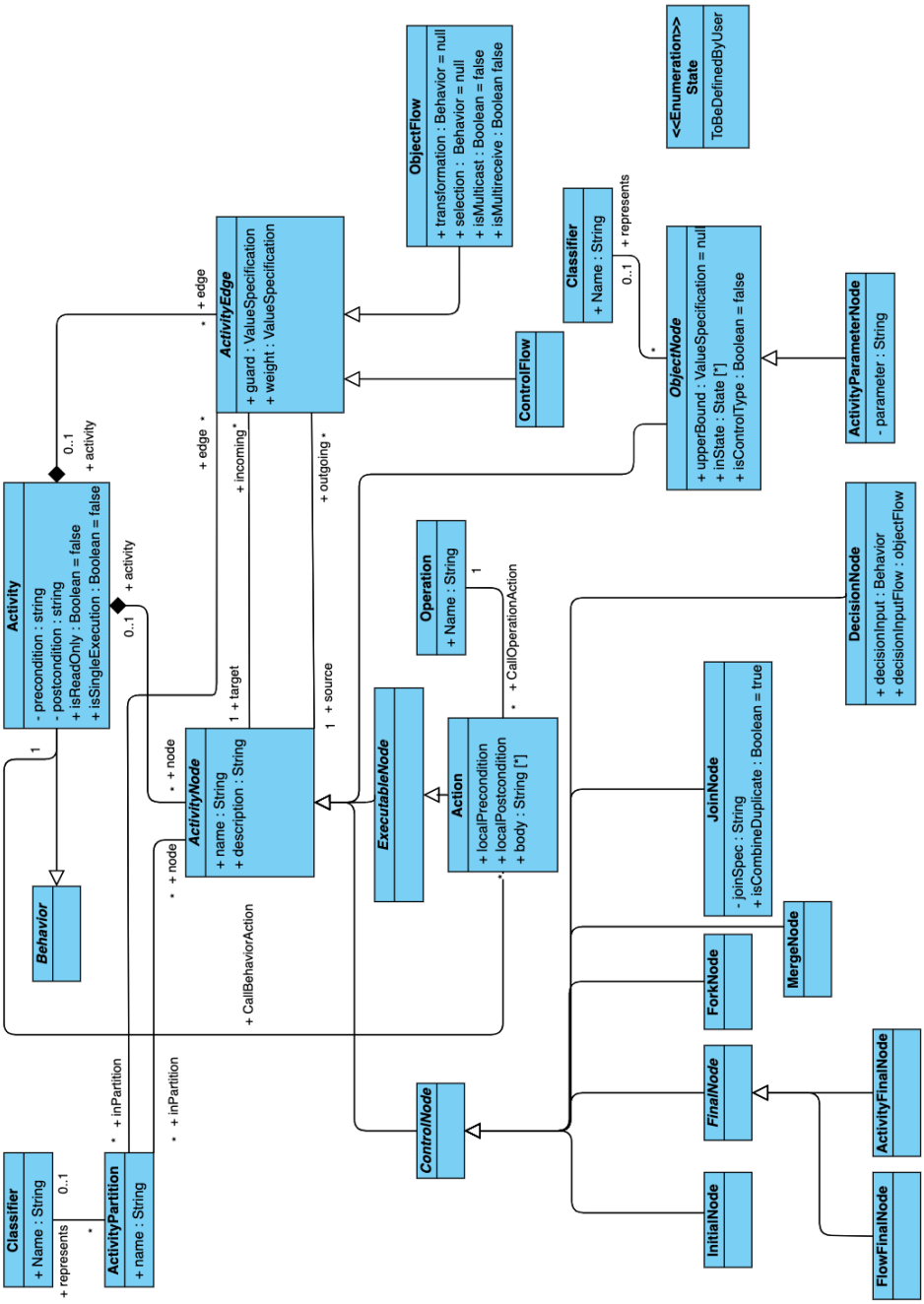

Figure 4.4: *Subset of Activity model classes.*

4.3.3.2 Behavior

Within UML there are different models to model certain objects, such as behavior. These models can be grouped into two groups, which are the static and behaviour models. The static models can model the static parts of a system, such as class models. Behaviour models are used to model the dynamic aspects of a system, such as how objects are changed over time. The behavior models are instantiated through the main behavioural class which is called *Behavior*. The class specifies the following instances: StateMachines, Activities and Interactions. These instances are used in Sequence and Activity models, which are called behavioural models. These models can model behavior and allow the behavior to be executed. As these behavioural models are part of the larger UML family it allows the different types of models to interact with one another, thus allowing classes to define activities. Through this connection, classes can specify the execution of behavior in the form of activities.

In our implementation, the Behavior class is tied to the Activity Class, such that other types of classes can make use of the behavior properties and methods. As shown in Figure 4.4 at the top the class Behavior is shown as a class from which Activity is inherited. This enables us to use other UML standards and refer to them through the Behavior class.

4.3.3.3 Action

Actions are the most basic form of activities in the Activity Model. The Action class models functionality that can be executed in some cases the functionality is behavior in itself. Through this mechanism, it is possible to point to another activity model. An example of this reference is the CallBehaviorAction, which is a subtype of the Action class. CallBehaviorAction points to another Activity model, that has a whole process defined. There are also other subtypes for the Action class, such as Invocation Actions, Object Actions, Variable Actions and more. The numerous subtype options for the Action class illustrate the large set of possibilities to model instances and behavior with the UML specification. On the other side, this large set of possibilities makes it complex to create simple representations. Therefore in our metamodel implementation, we have chosen to use a simplified set of some of these actions. In particular we make use of *CallBehaviorAction* and *CallOperationAction*, as these references are often used in Activity modelling practices.

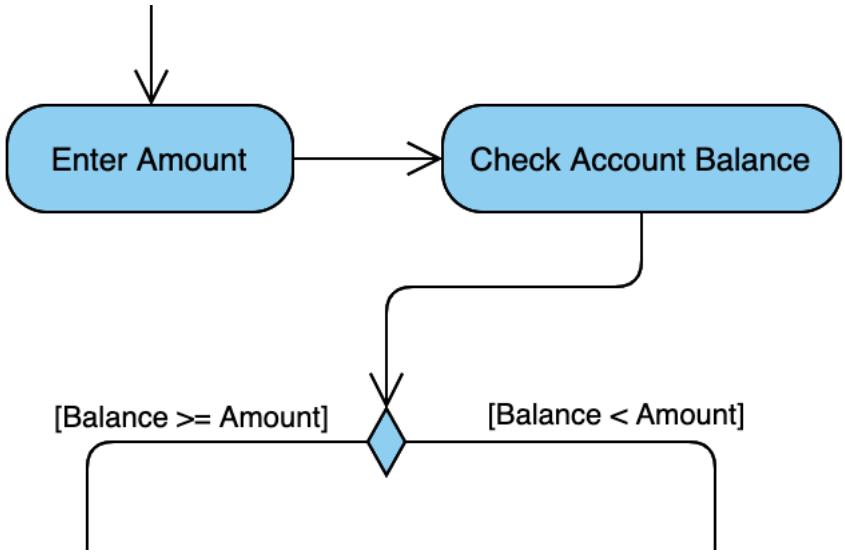

Figure 4.5: *ATM Activity Model example*

4.3.3.4 ValueSpecification

The class ValueSpecification can model a specification of a value for a class. It can have zero or more values, which can be of different types. The following types are directly integrated with a ValueSpecification: unlimitedNatural, string, real, integer and boolean. More types can be implemented in the ValueSpecification such as time, opaque, literal, interval and more. For now, we have implemented ValueSpecification as a string, such that multiple values can be added there. The interpretation of these values is for our approach currently not relevant, so that functionality can later be further specified.

The definition of a ValueSpecification does not give a clear meaning of how it works, therefore we show an example of a ValueSpecification. In the Activity model, the ValueSpecification is used in several classes, but for simplicity, we will use the guard attribute on an ActivityEdge. The guard attribute on an ActivityEdge is used to only allow tokens that evaluate true to a certain value, which is specified in the ValueSpecification. For example in Figure 2.1 of Booch et al. [7], which models the process of building a house. In this example, the edges after the decision node show a ValueSpecification, where the value of the token from the decision node can either be [not accepted] or [else]. As the diagram shows the process will continue if the value equals [else] otherwise it will return to the ”Bid plan” activity. In this case, the two ValueSpecifications are [not accepted] and [else], the values are strings and evaluated by the ValueSpecification.

The ValueSpecification can come in multiple shapes and sizes in Equation 4.1 and 4.2 we show some examples. These examples are from Figure 4.5 describing an ATM process. Where earlier activities have retrieved the Amount and Balance variables.

$$[Balance >= Amount] \tag{4.1}$$

$$[Balance < Amount] \tag{4.2}$$

With the definition of these variables, the next steps in the activity can be executed. We show the ValueSpecifications in the following equations: Equation 4.1 is used to evaluate if the card has enough funds to make the transaction and Equation 4.2 when the amount is insufficient. The example in Equation 4.1 and 4.2 are examples where two floats are compared and return a boolean from the comparison. Another common example in the ATM activity diagram is whether the pin code is valid. This is also a comparison that returns a boolean value.

4.3.4 UML editor

The UML editor is a modelling tool to model UML models. It is a web-based editor, which was developed by a student named Max Boone to create and edit UML class models. We have extended the capabilities of the editor by introducing the activity model in collaboration with Bram van Aggelen. In Figure 4.6 we show the editor with a class model.

With the introduction of the activity model, the editor can create and edit UML activity models. The editor is also able to save the model to a JSON file and to an external API which was described in Section 4.3.2. In Figure 4.7 we demonstrate an example of an activity model, that we generated with our pipeline. The added activity and action nodes are defined using the specified Metamodel from Subsection 4.3.3.

These examples are just some parts of what the editor can do. The editor is continuously in development. Other students are currently working on extending the modeller further, with several projects such as enforcement rules for the activity edges, a pipeline to pre-process requirements texts for the different NLP pipelines and more. The UML editor will continue to be further updated in the future.

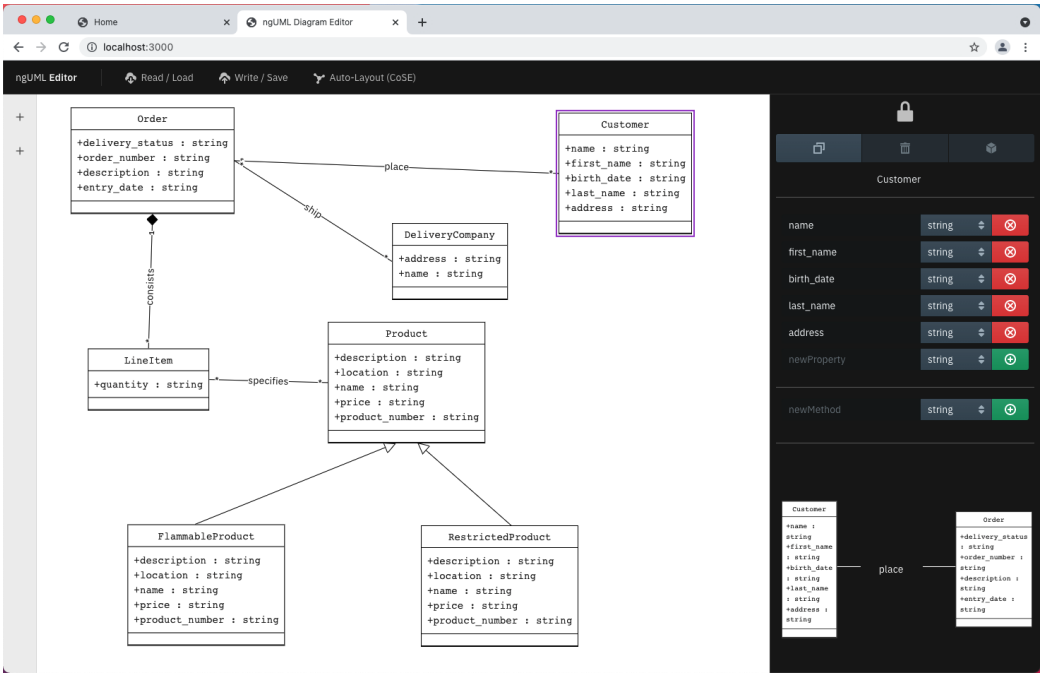

Figure 4.6: *UML editor with a class example*

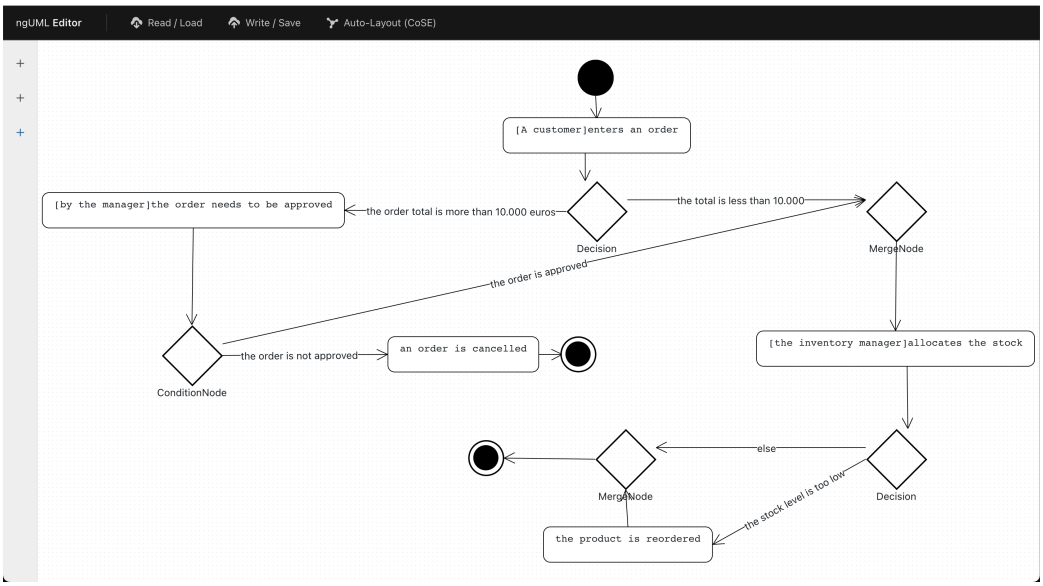

Figure 4.7: *UML editor with a generated activity model*

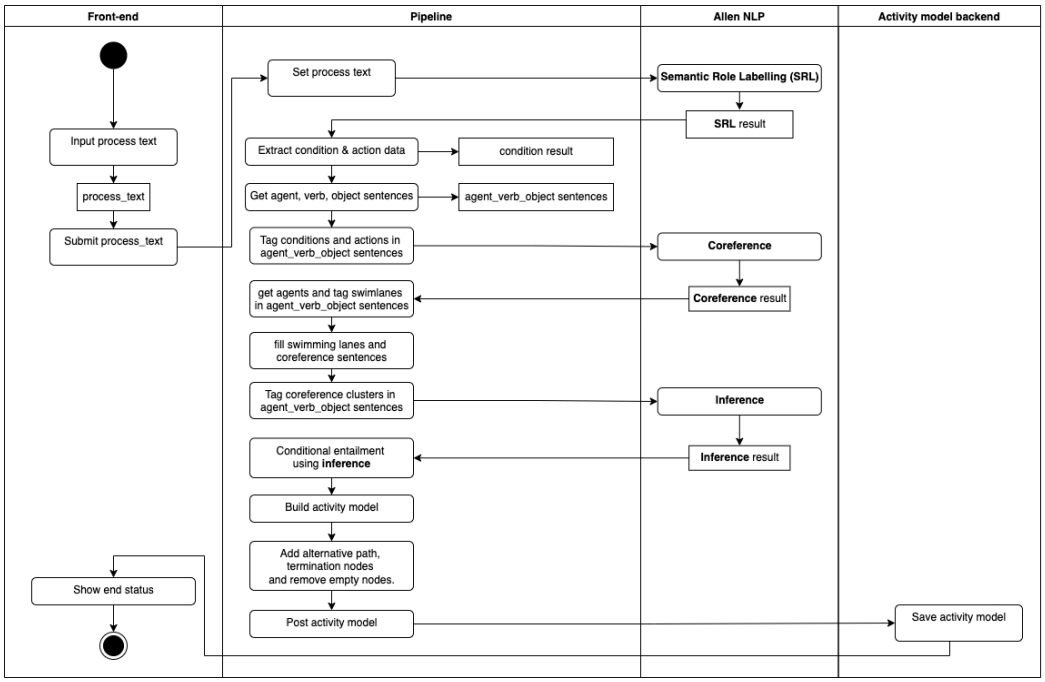

Figure 4.8: *Pipeline overview.*

4.4 In-depth overview of all actions

In this section, we once more show an overview of all the actions that occur in our pipeline. The difference with the overview in Section 3.1 is that in this overview we give more in-depth information on the different actions that are executed in our pipeline. We present this overview in our activity model in Figure 4.8.

Chapter 5

Analysis and Results

In this chapter, we will demonstrate the performance of our pipeline. We did consider to compare our results using precision and recall, but unfortunately, there is not a uniform dataset available to measure such performance for activity models. Therefore we will qualitatively compare the performance of our pipeline. We compare our generated models with a dataset of process descriptions and corresponding BPMN models. Next to that dataset, we also have some process descriptions without a business process model. For these process descriptions, we check if all actions and conditional structures are present in our generated model.

We have pushed 38 process descriptions through our pipeline to see if the pipeline does not fail. The used process descriptions were taken from the datasets in this chapter. We used 4 of these process descriptions more extensive, to see where our model could be improved. These four process descriptions have been taken from the dataset of the Humboldt University of Berlin, which were provided in the dataset of Friedrich et al.[32]. The titles of these process descriptions are the bicycle order, computer repair, underwriters and hotel process descriptions. We did not use these descriptions and models for the validation presented in this chapter.

For our validation approach we took four examples from the baseline dataset of Friedrich et al.[24], one process description from the FrameNet Requirements (FN-RE) dataset from Alhoshan et al. [30] and two process descriptions from the Public Requirements (PURE) Dataset from Ferrari et al.[31]. The models from the FN-RE and the PURE dataset do not have business process models with examples.

For each of the generated models, we will compare our model with that of a human or annotated text. For the models of Friedrich et al., we will also compare them to their generated model. We make use of red mark-

ings for the comparison in the model, these markings indicate the different problems with the model. In the text where we explain the differences, we further elaborate on what the problem is, why it occurred and what could be done to solve it.

Next to the comparisons with the requirements documents, we also present a collaboration with an industry partner. The goal of the collaboration is to explore the possibilities of connecting the NLP pipeline to an industry Business Process Modelling tool.

5.1 Research baseline

First, we will compare the results of our pipeline to that of a dataset that has been used in several research approaches as a baseline, that is the dataset of Friedrich et al.[24]. The dataset consists of 47 requirements texts and their corresponding BPMN model. The BPMN model comes in two varieties, one created by a human modeller and one generated by the NLP system of Friedrich et al.[24]. We will discuss four different examples, each with different properties, such as the source, length of the text and complexity of the text. We take the requirements text as input and generate an activity model, then we compare our model with the manually created model and the system-generated model from the dataset. In our development process, we have used a subset of the dataset for testing purposes. To make a clear distinction between validation and test data, we will not use these examples in our validation process. The texts and models we used for testing are the bicycle order, computer repair, underwriters and hotel process descriptions.

As previously mentioned the whole dataset consists of 47 requirements documents. Several of these documents have been translated from German into English and contain errors[33]. To overcome this problem and other typos, Bellan et al. have rewritten some of these texts to remove errors, therefore we make use of the versions from them[33]. Still, there could be more problems in the translations, to mitigate this risk we chose to not use the models that are of German origin. Next to the obstacle of translation, some models have been transformed from another model type (sequences) into a BPMN model. Sequence models are quite different from activity models, therefore we will also not use these models. With these considerations, the dataset is narrowed down to 21 models. From which we have selected 4. To demonstrate the properties of these models we show an overview in Table 5.1. We will discuss the results of each of these process descriptions in the next sections.

Name	Number of sentences	Source
Claim examination	5	Research
Service Level Agreement violation	38	Research
Loan approval - VOS	6	Business
Employee expense process - Oracle	15	Business

Table 5.1: *Overview of process descriptions from Friedrich et al. [32].*

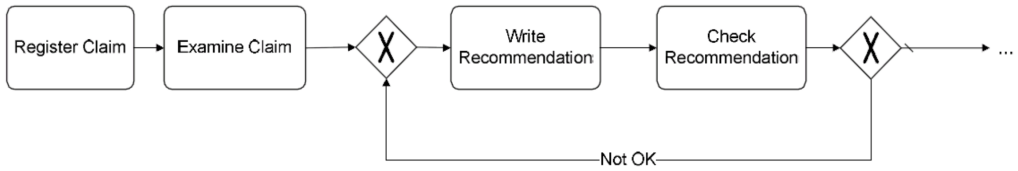

Figure 5.1: *Claim examination process, made by a human modeler.*

5.1.1 Repetition cycles - Claim examination

The claim examination process describes a process of a claim that is sent in. The description is presented in Example (1). Then in Figure 5.1 we present the BPMN model created by a human, the model created by our pipeline is shown in Figure 5.2 and the model from the system from Friedrich et al. is presented in Figure 5.3.

(1) *After a claim is registered, it is examined by a claims officer. The claims officer then writes a 'settlement recommendation'. This recommendation is then checked by a senior claims officer who may mark the claim as 'OK' or 'Not OK'. If the claim is marked as 'Not OK', it is sent back to the claims officer and the recommendation is repeated. If the claim is OK, the claim handling process proceeds.*
Claim examination process[24].

In Figure 5.2 we show our generated model with numbers to point to the problems of our model. The first comparison between our model and the manually created model shown in Figure 5.1, shows that our model has more information than the manually created model. The difference here can be explained through the approach of action creation. Some of the actions specified in the text area in the manual model are modelled through the structure of the model. On the contrary, we have specified these actions explicitly. To demonstrate such an explicit action we can have a look at the piece of text 'it is sent back to the claims officer'. Our model models this with the action 'a claim is sent back to the claims of-

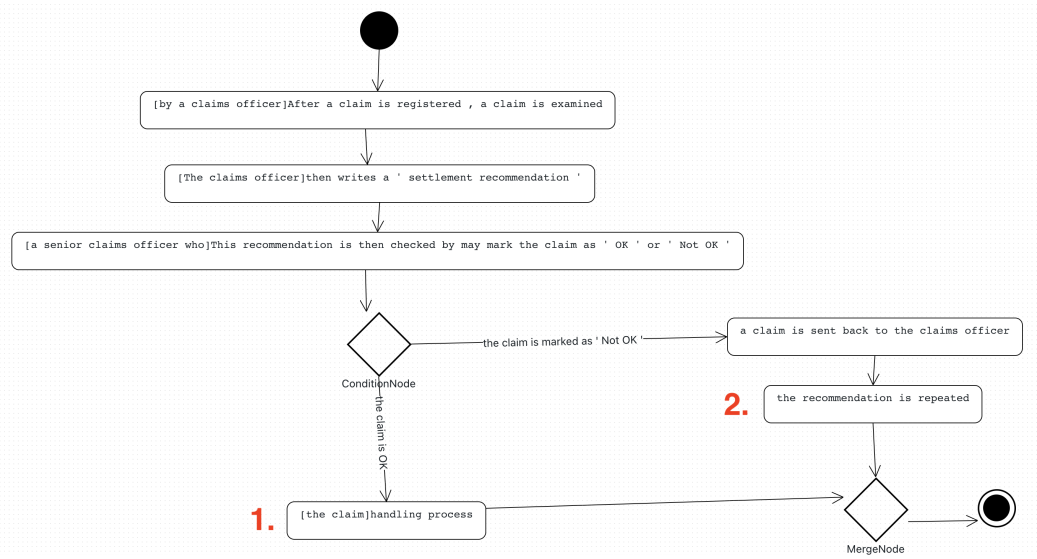

Figure 5.2: *Claim examination process with notes.*

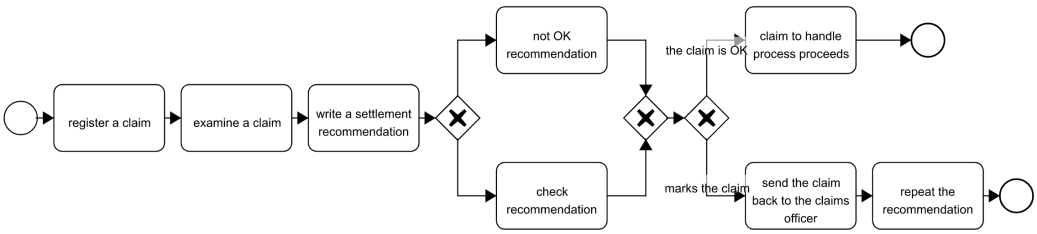

Figure 5.3: *Claim examination process, made by Friedrich system.*

ficer'. The manual model specifies this action with the edge and guards 'Not OK' that returns to the earlier gateway (decision). Another example of such a modelling structure versus an explicit action is the action that marks the claim as OK or Not OK. The manual model represents this with the gateway structure following the examine claim, our model shows this explicitly with an action.

Besides these differences, there are some errors with our model. The problem denoted with 1. in Figure 5.2 shows that the identified actor is not correct and the action text is incorrect. The actor is incorrectly predicted, because of our approach to actor identification. The approach extracts all ARG0 tags from an SRL roleset and states it as the actor, which is not per se the correct actor. The action should be 'the claim handling process proceeds', this is probably caused by the SRL roleset that was selected. The resulting roleset does not take the verb 'proceeds' into account with the found roleset.

The other problem with our model is the missing repetition denoted by the 2. in our model. This action should create a repetition, but as we previously mentioned in Section 3.5.2 we do not identify and create repetitions.

5.1.1.1 Difference generated model

The model in Figure 5.3 is generated by the pipeline of Friedrich et al.[24]. The models differ a bit. The model from Friedrich has an incorrect first conditional gateway because the 'not OK recommendation' action should follow the 'check recommendation' action. This also introduces a problem with the guards on the last gateway. Here the guard 'marks the claim' should be 'not OK recommendation'. In this comparison, the order of actions is better structured in the model from our pipeline. The actions from the model from Friedrich are better phrased but miss helpful context in some cases.

5.1.2 SLA violation

The SLA violation text is a process description of a Service Level Agreement violation. We collected it from the dataset of Friedrich et al., who took it from the TU Berlin[24]. It is the largest text we show in our examples.

(2) *At the beginning the customer perceives that her subscribed service has degraded. A list with all the problem parameters is then sent to the Cus-*

tomer Service department of TELECO. At the customer service an employee enters (based on the received data) a problem report into system T.. Then the problem report is compared to the customer SLA to identify what the extent and the details of the service degradation are. Based on this, the necessary counter measures are determined including their respective priorities. An electronic service then determines the significance of the customer based on information that has been collected during the history of the contractual relationship. In case the customer is premium, the process will link to an extra problem fix process (this process will not be detailed here). In case the customer is of certain significance which would affect the counter measures previously decided upon, the process goes back to re-prioritize these measures - otherwise the process continues. Taking together the information (i.e. contract commitment data + prioritized actions) a detailed problem report is created. The detailed problem report is then sent to Service Management. Service Management deals on a first level with violations of quality in services that are provided to customers. After receiving the detailed problem report, Service management investigates whether the problem is analyzable at the level of their department or whether the problem may be located at Resource Provisioning. In case Service Management assesses the problem to be not analyzable by themselves, the detailed problem report is sent out to Resource Provisioning. If Service Management is sure they can analyze it, they perform the analysis and based on the outcome they create a trouble report that indicates the type of problem. After Resource Provisioning receives the detailed problem report, it is checked whether there are any possible problems. If no problems are detected, a notification about the normal service execution is created. If a problem is detected this will be analyzed by Resource Provisioning and a trouble report is created. Either trouble report or the 'normal execution' notification will be included in a status report and sent back to Service Management. Service Management then prepares the final status report based on the received information. Subsequently it has to be determined what counter measures should be taken depending on the information in the final status report. Three alternative process paths may be taken. For the case that no problem was detected at all, the actual service performance is sent back to the Customer Service. For the case that minor corrective actions are required, Service Management will undertake corrective actions by themselves. Subsequently, the problem resolution report is created and then sent out to Customer Service. After sending, this process path of Service Management ends. For the case that automatic resource restoration from Resource Provisioning is required, Service Management must create a request for automatic resource restoration. This message is then sent to

Resource Provisioning. Resource Provisioning has been on-hold and waiting for a restoration request - but this must happen within 2 days after the status report was sent out, otherwise Resource Provisioning terminates the process. After the restoration request is received, all possible errors are tracked. Based on the tracked errors, all necessary corrective actions are undertaken by Resource Provisioning. Then a trouble-shooting report is created. This report is sent out to Service Management; then the process ends. The trouble-shooting report is received by Service Management and this information goes then into the creation of the problem resolution report just as described for ii). Customer Service either receives the actual service performance (if there was no problem) or the problem resolution report. Then, two concurrent activities are triggered, i.e. i) a report is created for the customer which details the current service performance and the resolution of the problem, and ii) an SLA violation rebate is reported to Billing & Collections who will adjust the billing. The report for the customer is sent out to her. After all three activities are completed the process ends within Customer Service. After the customer then receives the report about service performance and problem resolution from Customer Service, the process flow at the customer also ends.
SLA violation description[24].

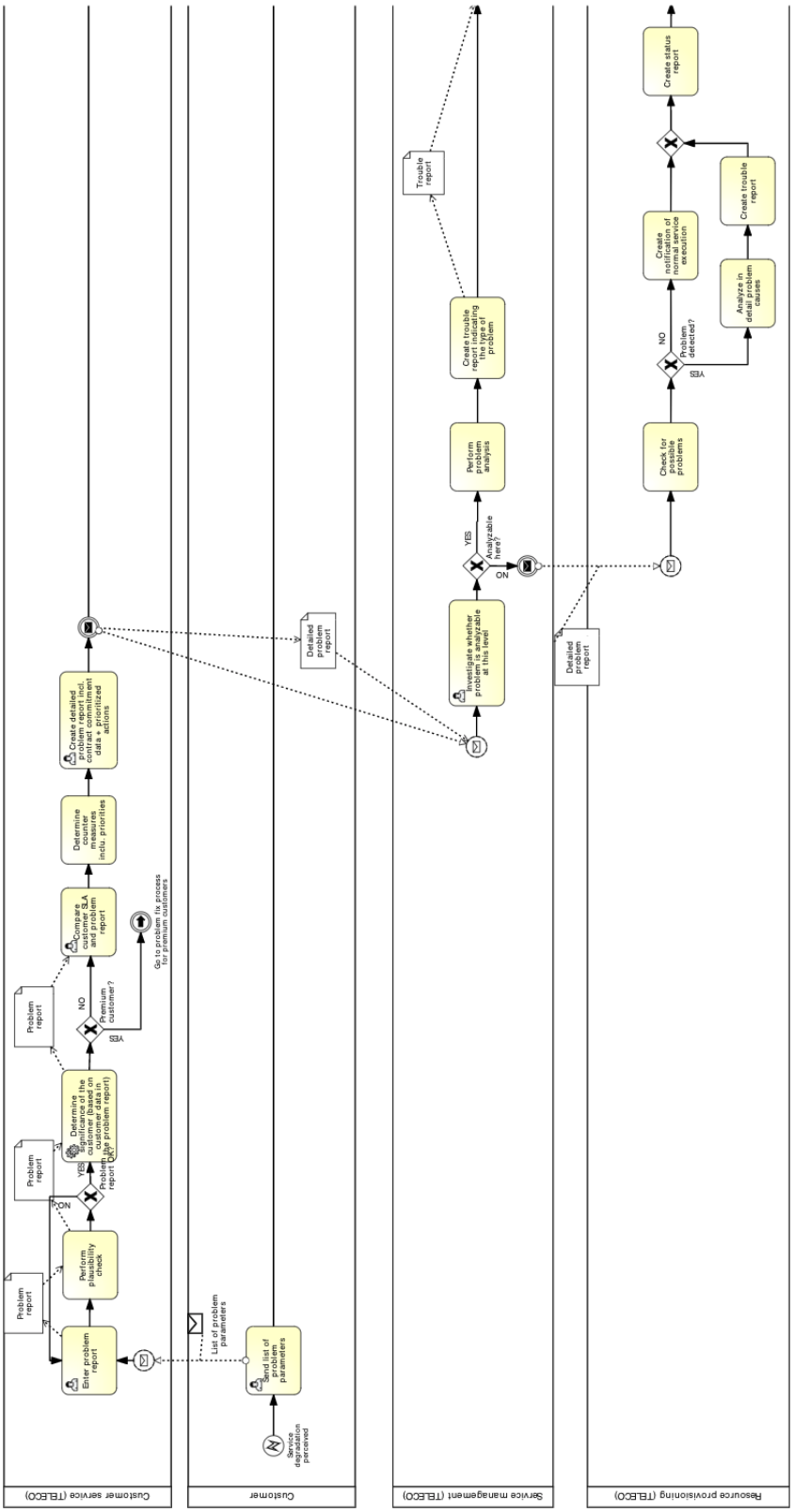

Figure 5.4: *SLA violation example generated by a human part 1[24].*

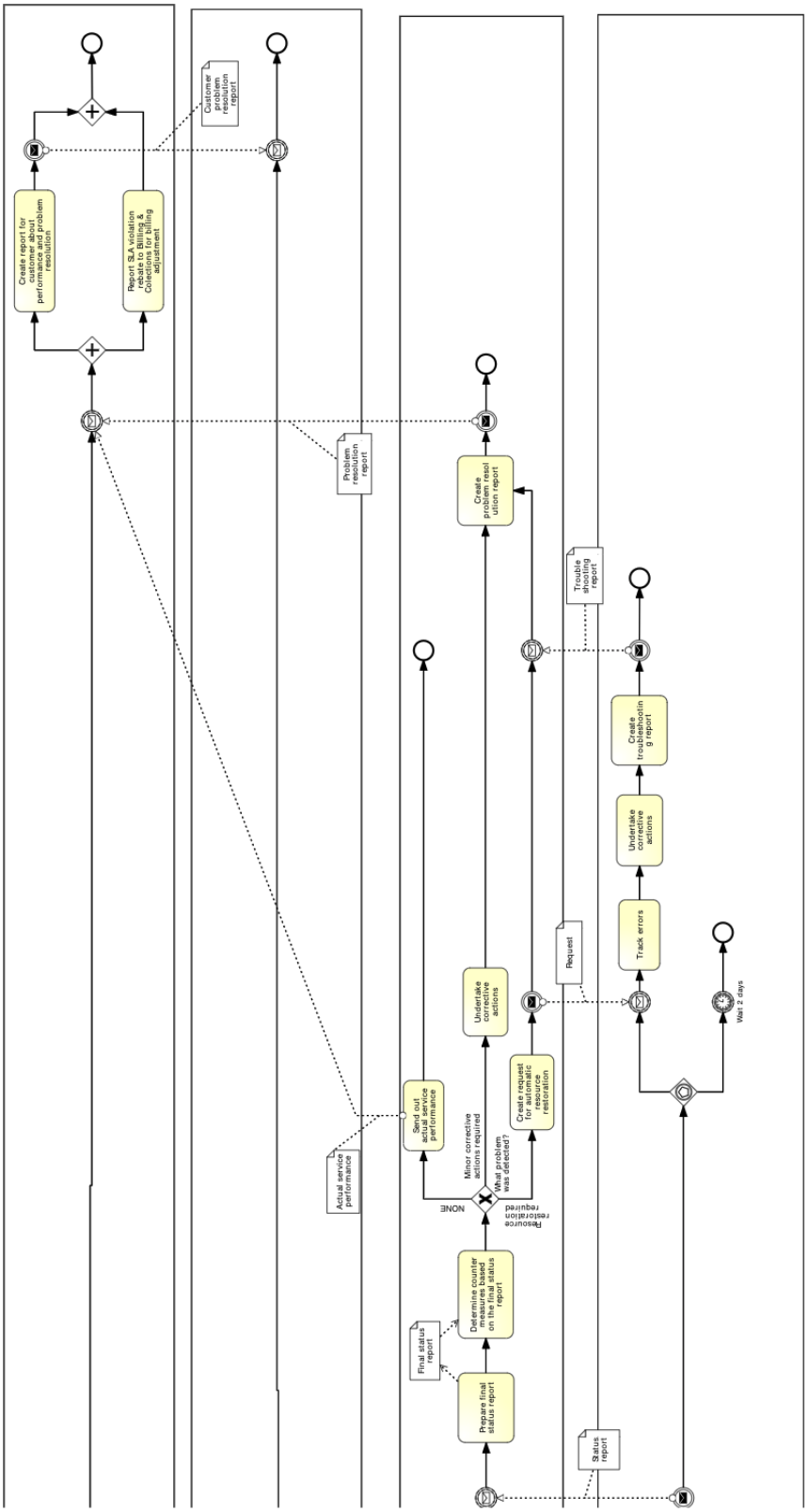

Figure 5.5: *SLA violation example generated by a human part 2[24].*

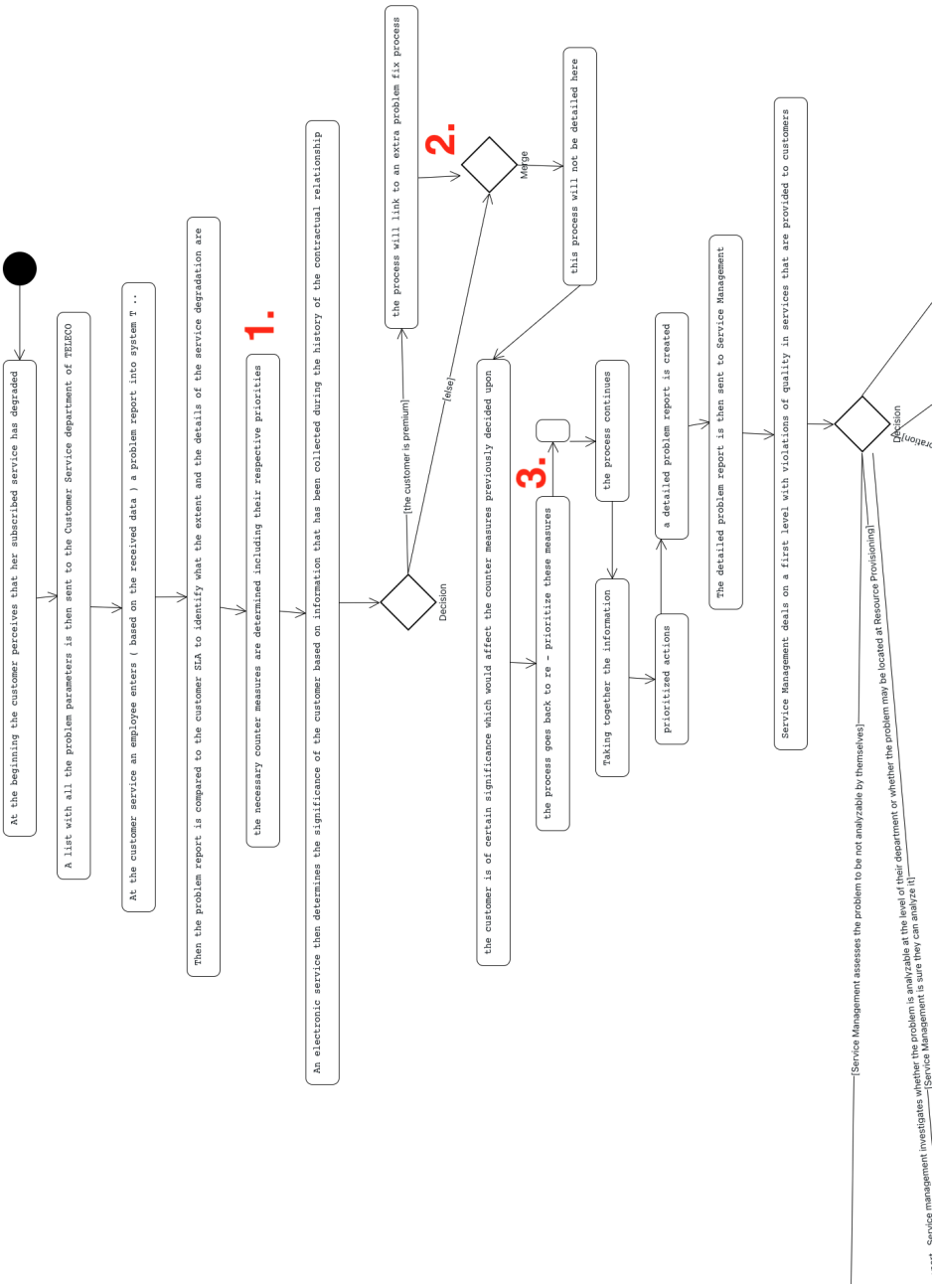

Figure 5.6: *SLA violation example generated by our system part 1.*

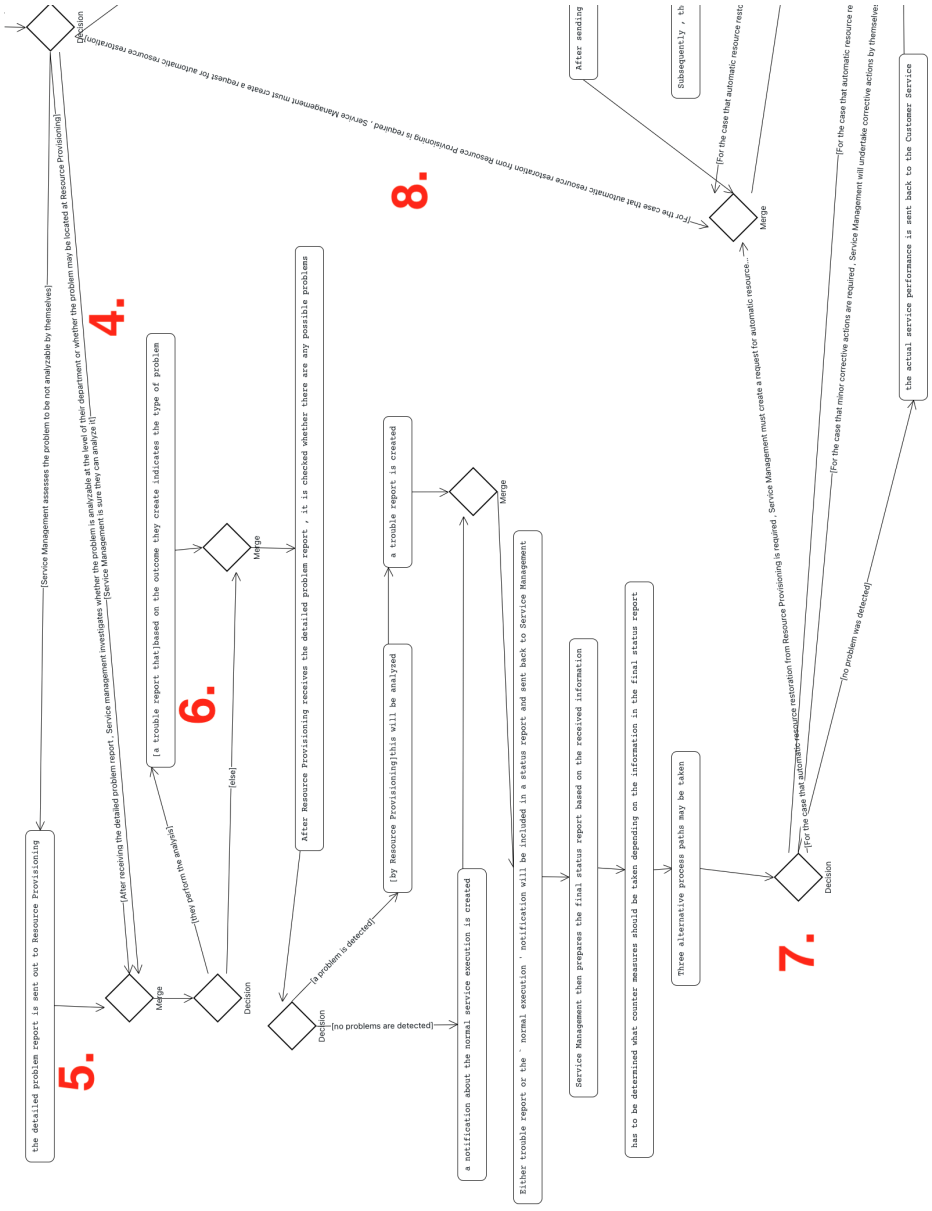

Figure 5.7: *SLA violation example generated by our system part 2.*

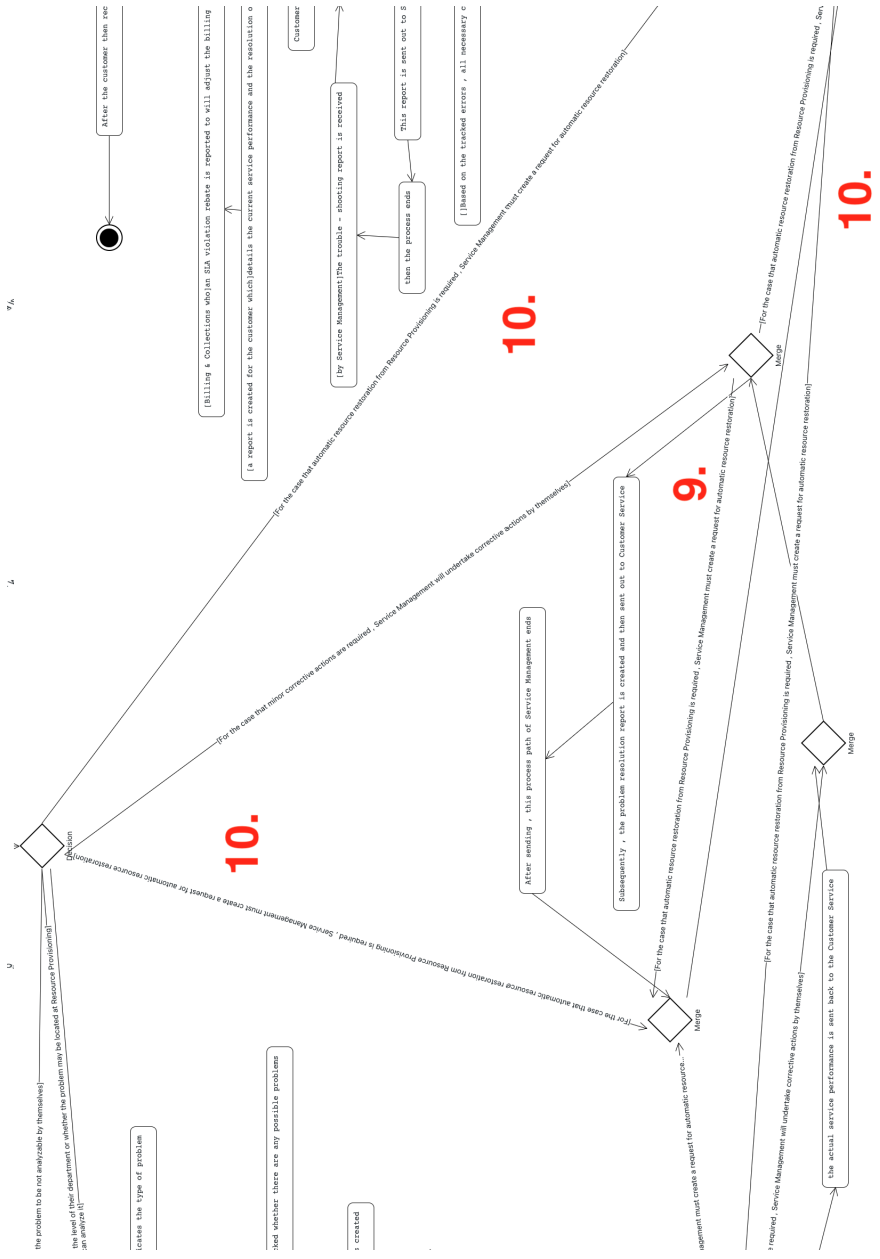

Figure 5.8: *SLA violation example generated by our system part 3.*

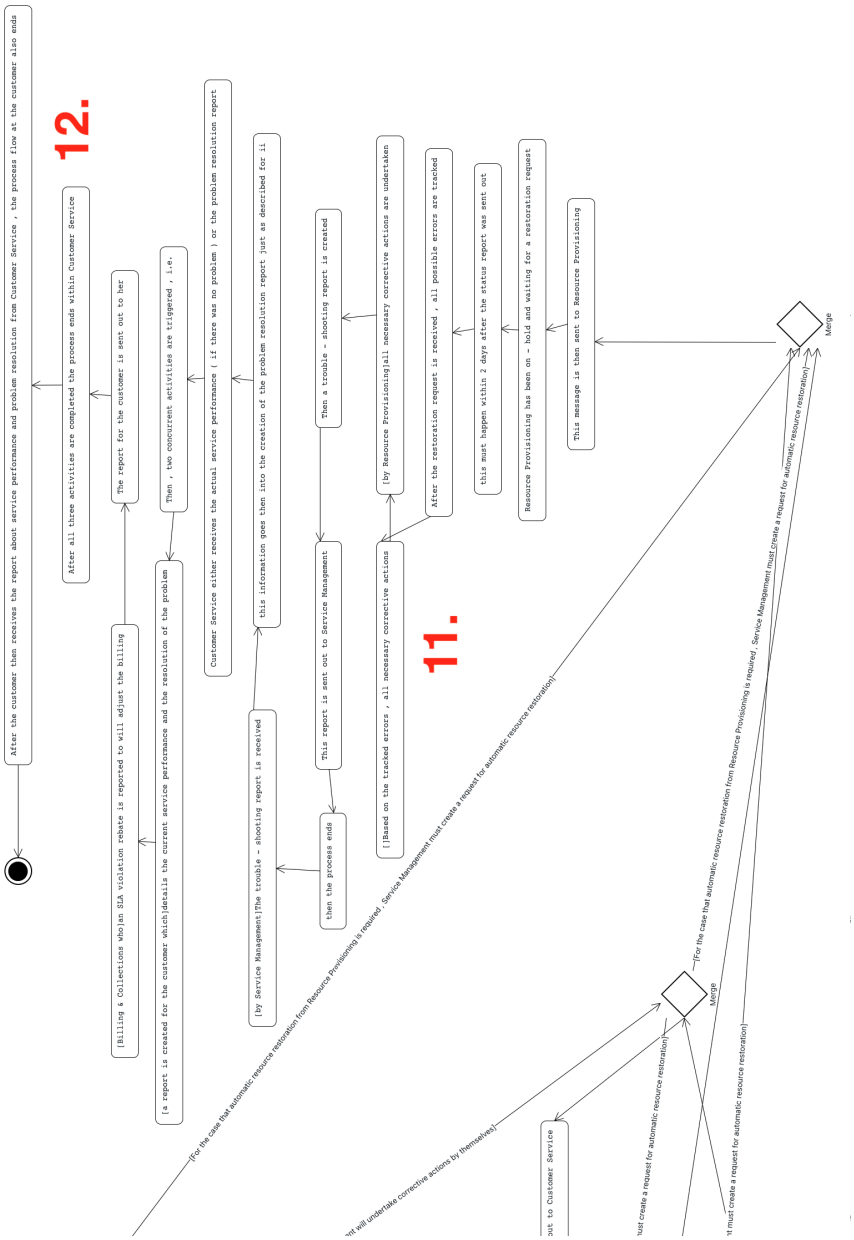

Figure 5.9: *SLA violation example generated by our system part 4.*

The generated model has been divided into 4 different screenshots because the models would not be readable if we put them all in one screenshot. The screenshots can be lined up using the different guards, that if you would put them next to one another would line up. We will start with Figure 5.10 and then mention if we go to the next screenshot.

Problem 1. points to an action that should have been a condition. The condition specified in the sentence modelled here is implicit, thus there is not a conditional keyword available. To process implicit conditions we would need to have some form of knowledge of the action to understand that it is a condition and be able to see how this would fit in the larger process model. Because of this required information, it is hard to process such implicit conditions.

The model from the human modeller shown in Figure 5.4 has at the first gateway (decision node) following the path for a premium customer, a link to another process. As mentioned in the text this process is not described in detail here, therefore we only know that it should link to another process. In our model at Problem 2. we model it as an action, this could be something to be improved, but we would need to understand where to end a process or go to another process. This would be a task for the sequence identification task in the text.

Problem 3. specifies a problem of a set of actions that could have been a conditional structure. These actions are part of a sentence that starts with the conditional keyword 'in case of'. Our model does not handle this correctly which is similar to the model in Figure 5.4 where the sentence was also created as two actions instead of a conditional structure. The actions in Figure 5.4 that specify this are *'compare customer SLA and problem report'* and *'Determine counter measures inclu. priorities'*. Our model also was not able to process this as a conditional structure, while it should have been because we filter for conditions using the keyword 'in case'. This is probably an error. Another problem with our model is the empty action. We suspect this empty action to be created because of the character '-' in the sentence and the SRL roleset around that character. These characters might also have been the reason for the problem with the missing conditional structure.

After these actions, we have several actions that could have been combined into one action. Starting with the action *'Taking together information'* and the last action *'a detailed report is created'*. Our model split these because we create actions based on SRL rolesets, in the case of this sentence, there are multiple rolesets. The actions can be combined by a UML modeller.

Then we continue with the next part of the model in Figure 5.11. Here Problem 4. shows an incorrect guard, that we present in Example (3).

(3) 'After receiving the detailed problem report, Service management investigates whether the problem is analyzable at the level of their department or whether the problem may be located at Resource Provisioning'.

This part of the text should have been created as an action such as in the human model: *'investigate whether this problem is analyzable at this level'*, then after this action there should be a decision node with the choice between analyzable by them or the service provisioning department. The model that we have generated makes such a decision structure with the other two guards: *'Service management assesses the problem to be not analyzable by themselves'* and *'Service management is sure they can analyze it'*. So only the guard that should have been an action is problematic in that structure. The error of our model is caused by the conditional keyword 'whether' that is within the part of the text. That makes our pipeline process it as a condition, and because of a conditional structure it is coupled with the guard *'Service Management assesses the problem to be not analyzable by themselves'* and our entailment predicts it as a contradiction. To solve this we would narrow down our conditional keywords, but there are cases where we should keep the word 'whether' as a condition. So it is not an easy to fix solution.

For Problem 5. we are dealing with a correct condition and action, but the process that occurs after the last action should be continued at another merge node. This problem is the effect of Problem 6. because the flow should continue at the Resource Provisioning actions. Problem 6. does not make that possible. Then Problem 6. shows an incorrect conditional structure. The guard *'they perform the analysis'* should be an action instead of a guard. What is strange about this guard is that there is not a conditional keyword present. This guard is the result of our assumption that if a conditional keyword is in the sentence and we have an adverbial, the adverbial part of the sentence specifies the condition. In Example (4) we present the sentence.

(4) 'If Service Management is sure they can analyze it, they perform the analysis and based on the outcome they create a trouble report that indicates the type of problem.'

In this sentence, the first adverbial is *'If Service Management is sure they can analyze it'*, which is the correct condition. However in this sentence, there is another adverbial: *'based on the outcome'*, this caused the pipeline to make the range of our adverbial also include the other adverbial. We use the adverbial range to find out if a found action is part of a condition.

We specify the adverbial range by checking for the adverbial's begin and end tag in an SRL roleset. The range identification is fairly straightforward and does not take into account that an adverbial in one SRL roleset can be split. Thus the action *'they perform the analysis'* is taken by our pipeline as part of the adverbial that specifies a condition. If the action was not tagged as a condition it would have been an action and part of the alternative path with the guard *'Service management is sure they can analyze it'*. To solve this we will need to be more specific with the identification of our condition and adverbials.

As mentioned with Problem 5. we should have continued after the merge node from Problem 6.. Besides these problems, we also see a clear difference with the human-generated model in Figure 5.4, the use of swimlanes. Our pipeline does not generate such swimlanes, because we did not implement it and there are multiple approaches to it. Nevertheless, it is important for the structure of a model, especially in cases where the actions of each actor are not following one another. In most examples, we have seen the actions directly follow one another and actions per actor are clustered together. Still, there are texts where this is not the case, in those cases, swimlanes are a good addition to the structure and an important aspect for future work.

Then we come to Problem 7., here we have a decision node with correct and incorrect guards. We present the three guards in Example (5).

(5) Three guards near Problem 7.

a. 'no problem was detected'
b. 'For the case that minor corrective actions are required, Service Management will undertake corrective actions by themselves'
c. 'For the case that automatic resource restoration from Resource Provisioning is required, Service Management must create a request for automatic resource restoration'

The guard in Example (5-a) is correct with its' action that follows, different from the model made by the human is the fact that we do not stop the process. The next guard is Example (5-b), this should only have been *'the case that minor corrective actions are required'*. We looked into the SRL rolesets and saw that there is another ARGM tag used for the part of the condition than the adverbial tag that we have used. The condition is tagged as an ARGM-PNC, which is an argument purpose not cause. The tag specifies the motivation for a certain action. This led to a roleset that filled the whole sentence and as it has a conditional keyword the whole sentence became a condition. Then there is a set of actions that should have been linked to

this guard, which we will discuss later as we first discuss the other guard at the current decision node. The third guard is Example (5-c), here we have the same problem as with the guard in Example (5-b), because the condition is marked with the SRL tag ARGM-PNC. So it should have been split into a guard and action node.

Besides the problem with the guard itself, it is good to see that we can link this guard to the decision node that was created three actions back in the text. The link is possible through the use of coreference clusters to link conditions. A side effect (Problem 8.) here is the fact that this and other guards have been linked to several other decision nodes that are in the same coreference cluster and have one contradiction through the comparison with entailment. This approach of multiple incorrect guards could have been solved by only creating one coreference cluster guard and first checking for the closest decision points from the text.

Now we move onto the next screenshot of the model in Figure 5.8. We start with Problem 9. here we see the two actions that should have followed the guard with the *'minor corrective actions are required'*. The actions themselves are correctly stated when we look at the model created by a human, only the path they are on is incorrect. To combine these actions with the condition we would need to be able to couple them. For the standard conditions we have used the sentence structure, but these actions are not part of the sentence. So this is hard to solve, it could be done by identifying possible follow-up sentences through entailment, which we have tried. Unfortunately, the problem with entailment is that many sentences can be seen as an entailment for another sentence, but they might not be the correct following sentence. The entailment approach is not very strict.

Then Problem 10. which was already specified, but we would like to show it again. Through the use of coreference to find similar conditions and compare them with entailment, several guards have been created. As we previously mentioned with Problem 8. that the solution lies in limiting the number of coreference guards the pipeline creates. In such a solution we only create one guard for a coreference guard and use a specific search solution to consider decision node candidates. The search approach will first select the closest conditional sentences and extend from there, to make sure we find the best solution first.

We continue to the final screenshot with Figure 5.9. Here we have the final set of actions that are part of this activity model. The first problem we encounter is 11. this problem indicates a missing actor. We are not sure why the actor is not found, because the ARG1 tag is present in the role set and in the action we suggest that this has to do with the overlap with the next action.

Finally, the last problem 12. is the missing of a parallel path. In our approach we have not implemented parallel paths, thus it is not a problem, but it could be implemented in the future.

To conclude our comparison, most of the actions and structures are modelled by our generated model. Not all, but many actions that have been presented in the human-generated model are also present in our model. Next to that most simple conditional structures are also generated by our model. The more complex structures are partially created which is the effect of implicit conditions or not correctly modelled conditions by our approaches. Nevertheless, the generated model does present a good first approach to generating models and a good starting point for the creation of an activity model.

5.1.2.1 Difference with generated models

In comparison with the model of Friedrich et al. in Figure 5.10 and 5.11, we see differences on multiple levels[32]. They have several actions that have been put to the active form, which makes it easy to read and the actions are clear.

A disadvantage of the approach to converting text and rewriting the text to create actions is that in some cases information is lost. Two examples of this are actions that are removed and actions where important words were removed. One removed action is the final action *'receive the problem resolution'* that is removed because of extraction. An example of important words missing is the action *'perform based'*, which misses information on what it should perform based on.

Furthermore, there are problems with multiple conditional structures. The second condition after the action *'fix process'* is a conditional statement that misses the guards and misses correct actions. We have shown with Problem 3. that our pipeline was also not able to create this condition correctly. The next problem with a conditional structure is at the guard *'the problem is analyzable'*. This part of the text is also quite hard to correctly process, as we also see in our model, that we demonstrate at Problem 4..

The next problem is the next conditional structure here we see that the guard *'service management is sure'* is incorrect. The rewriting of the text removed the part of the sentence that explained the guard, which is *'they can analyze it'*. With the use of our models' complete sentences we kept the context here and therefore the correct guard. Another guard problem is at the next conditional structure. Here we have the guards define a problem detection, the guards about detection are correct, but the guard *'possible problems'* should not have been a separate path. It should have

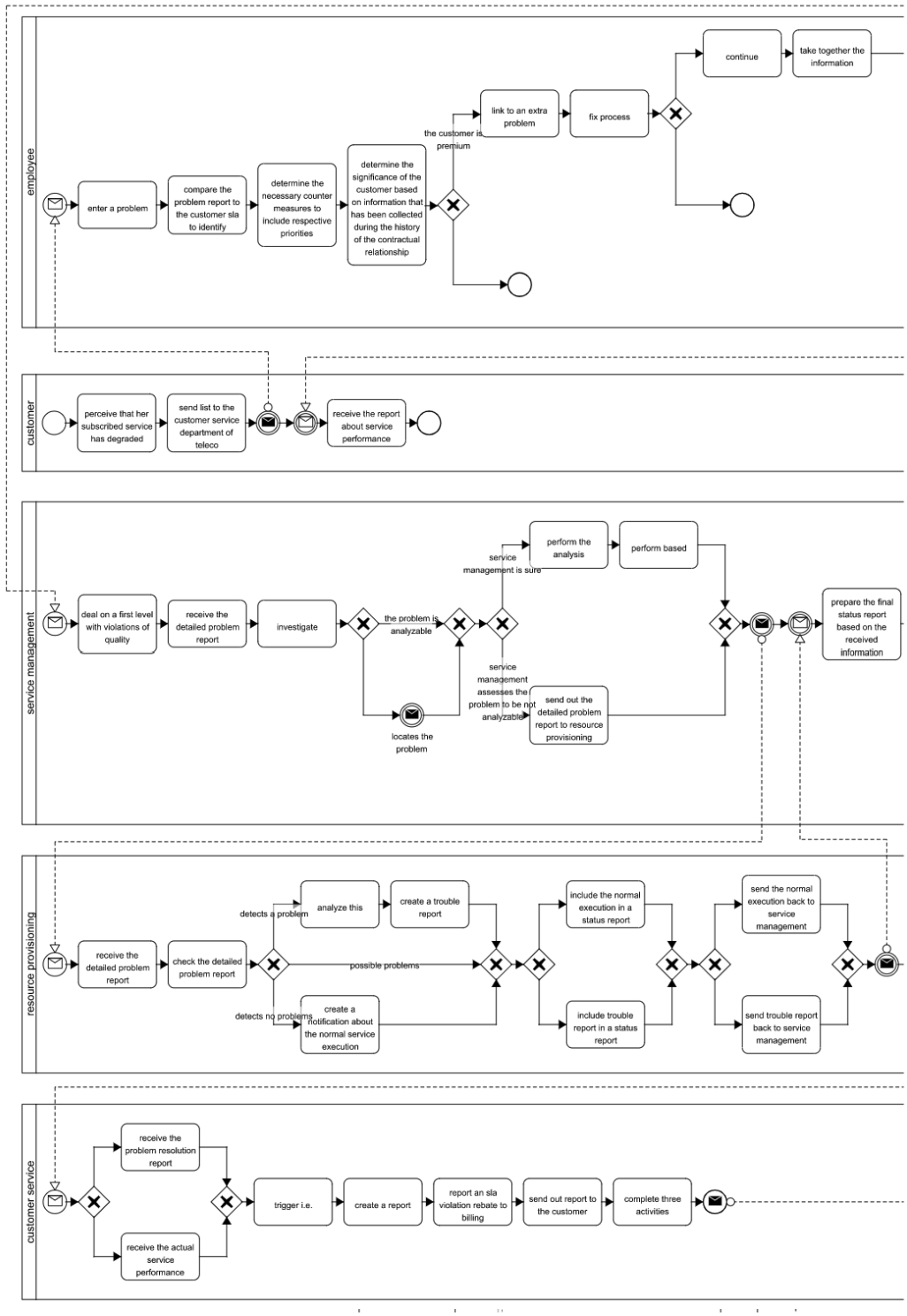

Figure 5.10: *SLA violation example generated by a system part 1[24].*

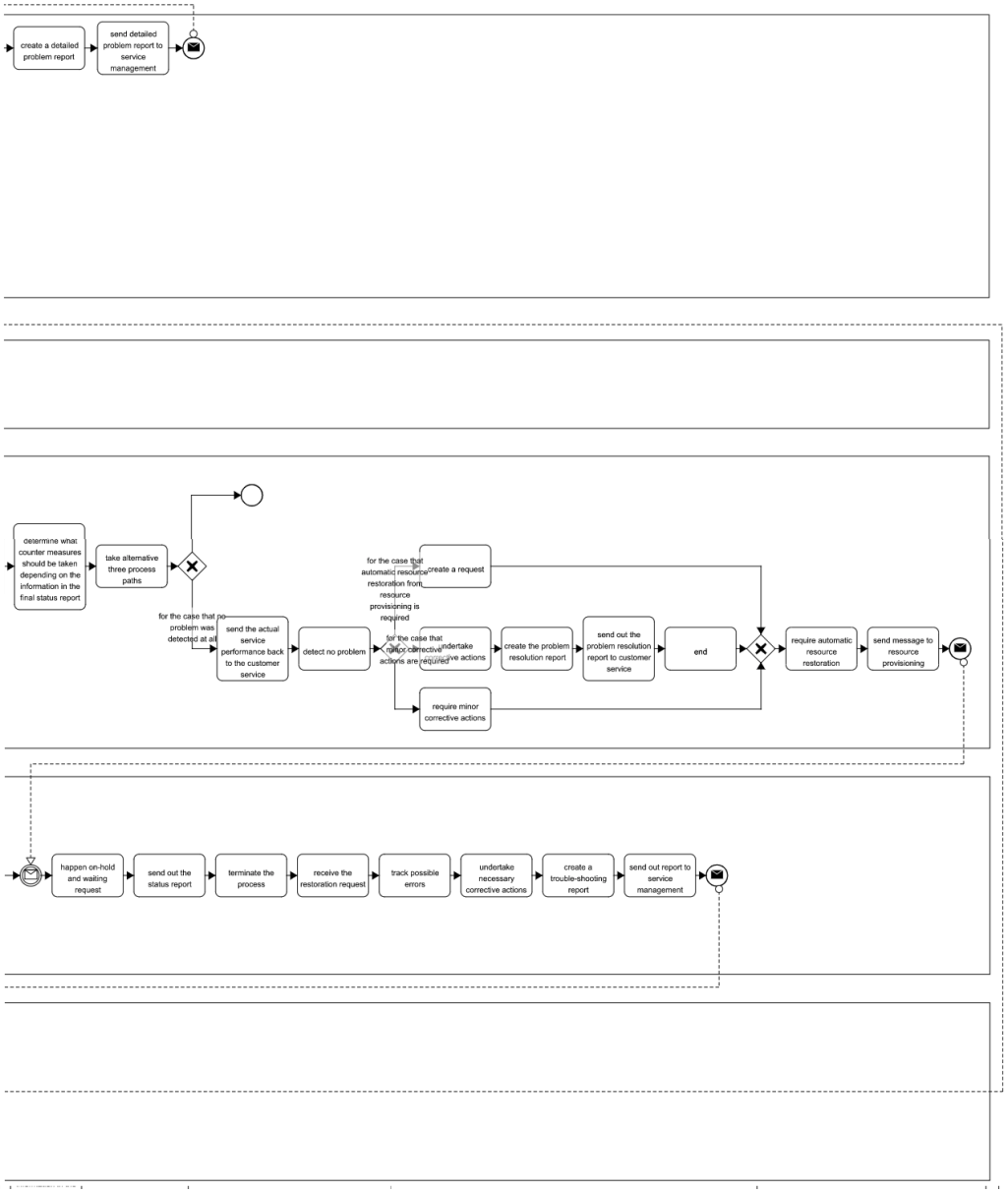

Figure 5.11: *SLA violation example generated by a system part 2[24].*

been an action before the first conditional node. Also, some important information is removed with the action of investigating whether there are possible problems. Our model did not model the part of the sentence as a condition, because there was not an action following after the condition.

Furthermore, there are several conditional structures without any guards, but these are correct because in the text there is not a condition given just two different paths.

Then the next conditional structures with guards are not combined correctly and there is even a process termination. The three possible paths are correctly found, but they are not linked together. Next to that, there are incorrect actions. Our model can correctly combine these paths in the same node, but it also introduces problems with too many edges with the same guard. We know how to fix this, thus it shows that the approach of using coreference and entailment enables us to create more correct and complex conditional structures. One thing that the system of Friedrich et al. does better in this conditional structure is the identification of actions that belong on a certain path. The actions following the guard *'for the case that minor corrective actions are required'* are the actions that should be part of this alternative path. In our model we were not able to do this correctly, we demonstrate this with Problem 9.. There are more points to discuss, but we would like to keep our analysis to this point.

We do see that the model of Friedrich et al. does model the different processes quite correctly. Especially it is good at defining concise actions and is good at finding certain paths of actions. The conditional structures can be improved and we have shown a way to do this. The problems of Friedrich et al. are not to show that is a bad solution, because it is a really good solution and in some ways still outperforms our solution. We want to show what is going wrong, to identify where we could improve. Our pipeline also needs improvement in certain areas, but it does create a first good starting point for the model generation.

5.1.3 Loan approval process - VOS

The loan approval process is a process description from a tutorial on modelling a description as a business process. The description is from a tutorial with the BPMN tool ActiveVOS.

(6) *The loan approval process starts by receiving a customer request for a loan amount. The risk assessment Web service is invoked to assess request. If the loan is small and the customer is low risk, the loan is approved. If the customer is high risk, the loan is denied. If the customer needs further*

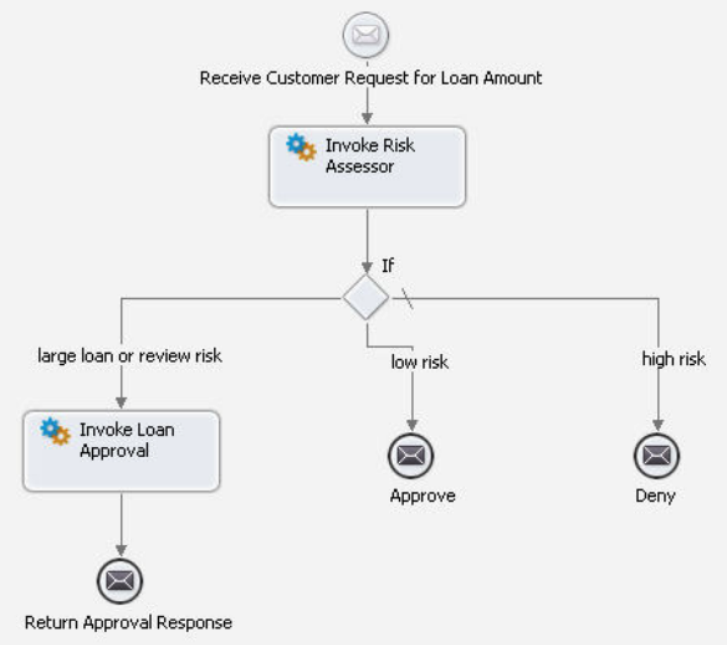

Figure 5.12: *Loan approval process, made by a human modeler.*

> *review or the loan amount is for $10,000 or more, the request is sent to the approver Web service. The customer receives feedback the assessor or approver.*
> Loan approval process - VOS [32].

In Figure 5.13 we demonstrate the model that resulted from our pipeline with the process description as input. There are some problems with our model. Problem 1. specifies a problem with the conditional statement. The condition in the text is *'the loan is small and the customer is low risk'*, in our model the statement has been divided into two different conditions. For Problem 2. it is an arguable problem, in the model in Figure 5.12 we see that after the *'high risk'*, *'low risk'* or *'review'* guard there is a mail icon. This icon represents the feedback that is sent to the customer. In our model, this action is also executed at the end of the whole flow of actions. One could argue this should have been directly after the actions *'the loan is denied'* or *'the loan is approved'*. On the contrary, the following guards and actions can just be ignored, because these will not be executed as we follow the flows with *'[else]'*. Thus the problem here is a bit of a style and best practices discussion, because of the complexity of path identification we were not able to merge and combine these actions in the mentioned way.

The path identification does bring us to the final problem, that we have

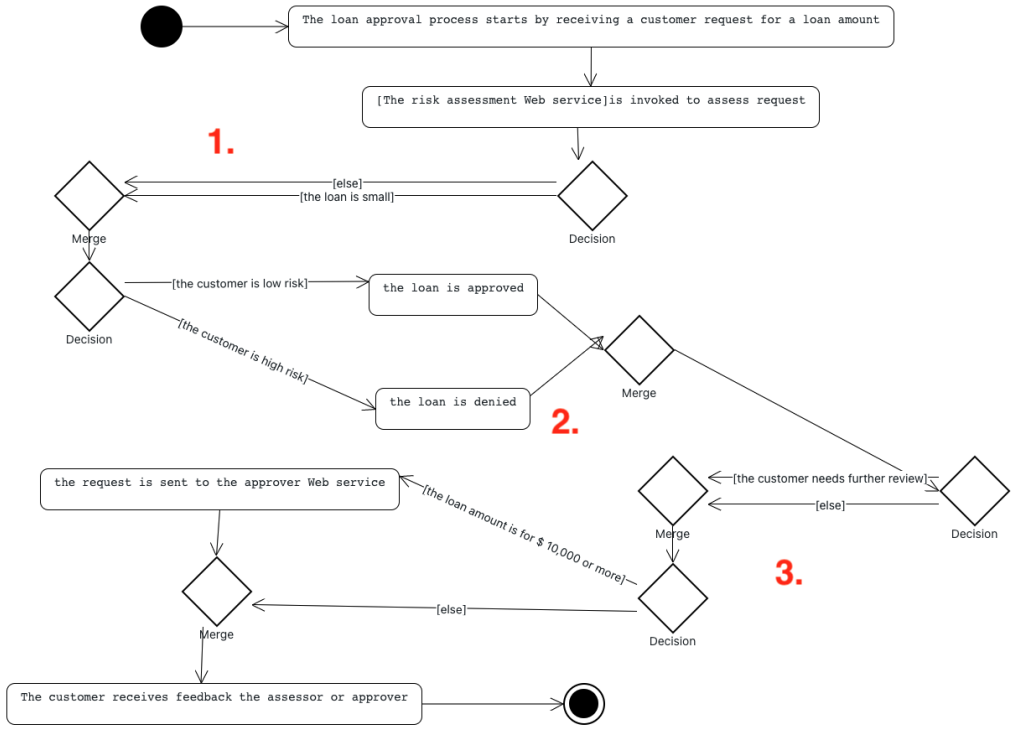

Figure 5.13: *Loan approval process with notes.*

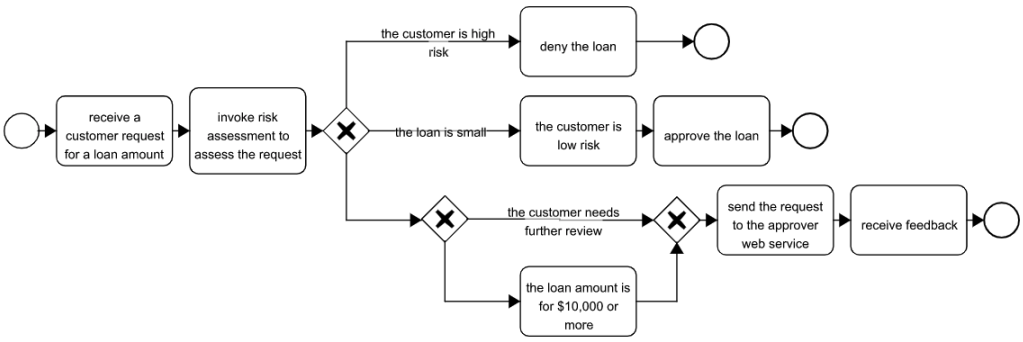

Figure 5.14: *Loan approval process, made by Friedrich system.*

denoted with Problem 3. At first, glance we see a similar problem with the two conditions being split up. This is a result of the way we process conditions because there are two ways to handle them. We could keep them as separate conditions which might help us in linking back to earlier conditions or always merge them. In hindsight, we should have merged the two separate conditions, if we could not couple them to an earlier condition (through coreference). Besides this division of conditions, we would like to have seen the path that starts at *'the customer needs further review'* to be another path next to the guards *'the loan is small or the customer is low risk'* and *'the customer is high risk'*. Our approach to combining such conditions that are not directly following one another is a combination of coreference clusters and entailment. The coreference cluster allows us to compare conditions that are apart in the text. The entailment enables us to compare the sentences on a semantic level. We only state that a condition is an alternative condition from another condition if the highest prediction of the entailment model returns 'contradiction', which means that the conditions contradict one another. In this example, we did compare the guard *'the customer needs further review'* with the *'the customer is low risk'* and *'the customer is high risk'*, because they have the same coreference cluster of 'customer'. Unfortunately, the guard *'the customer needs further review'* is not a contradiction of the other two conditions, thus we do not consider the condition as an alternative path.

5.1.3.1 Difference generated model

In comparison with the model generated by Friedrich et al. in Figure 5.14[24]. They also have the guard of *'the loan is small'* not combined with *'the customer is low risk'*. The same is the case for the review guard and the amount of the loan. The action of the customer receiving feedback is incorrectly shown because the customer is not defined. Finally, the action of receiving feedback should have been a final step for all the paths. Here we also see that it is hard to define a 'correct' condition and identify the paths in a logical order.

5.1.4 Employee expenses process - Oracle

The text from the Oracle tutorial specifies the process of processing a list of employee expenses. We present the textual description in Example (7).

(7) *An employee purchases a product or service he requires. For instance, a sales person on a trip rents a car. The employee submits an expense report*

with a list of items, along with the receipts for each item. A supervisor reviews the expense report and approves or rejects the report. Since the company has expense rules, there are circumstances where the supervisor can accept or reject the report upon first inspection. These rules could be automated, to reduce the workload on the supervisor. If the supervisor rejects the report, the employee, who submitted it, is given a chance to edit it, for example to correct errors or better describe an expense. If the supervisor approves the report, it goes to the treasurer. The treasurer checks that all the receipts have been submitted and match the items on the list. If all is in order, the treasurer accepts the expenses for processing (including, e.g. , payment or refund, and accounting). If receipts are missing or do not match the report, he sends it back to the employee. If a report returns to the employee for corrections, it must again go to a supervisor, even if the supervisor previously approved the report. If the treasurer accepts the expenses for processing, the report moves to an automatic activity that links to a payment system. The process waits for the payment confirmation. After the payment is confirmed, the process ends.
Oracle tutorial text[32].

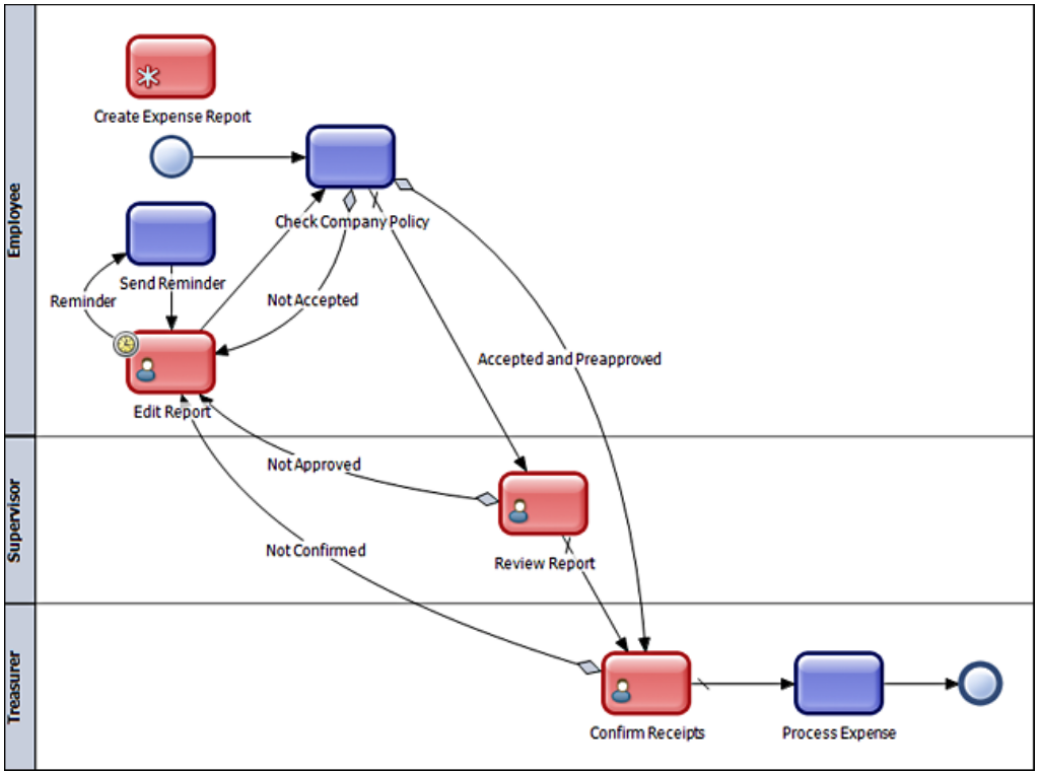

Figure 5.15: *Oracle expense processing example made by a human[24].*

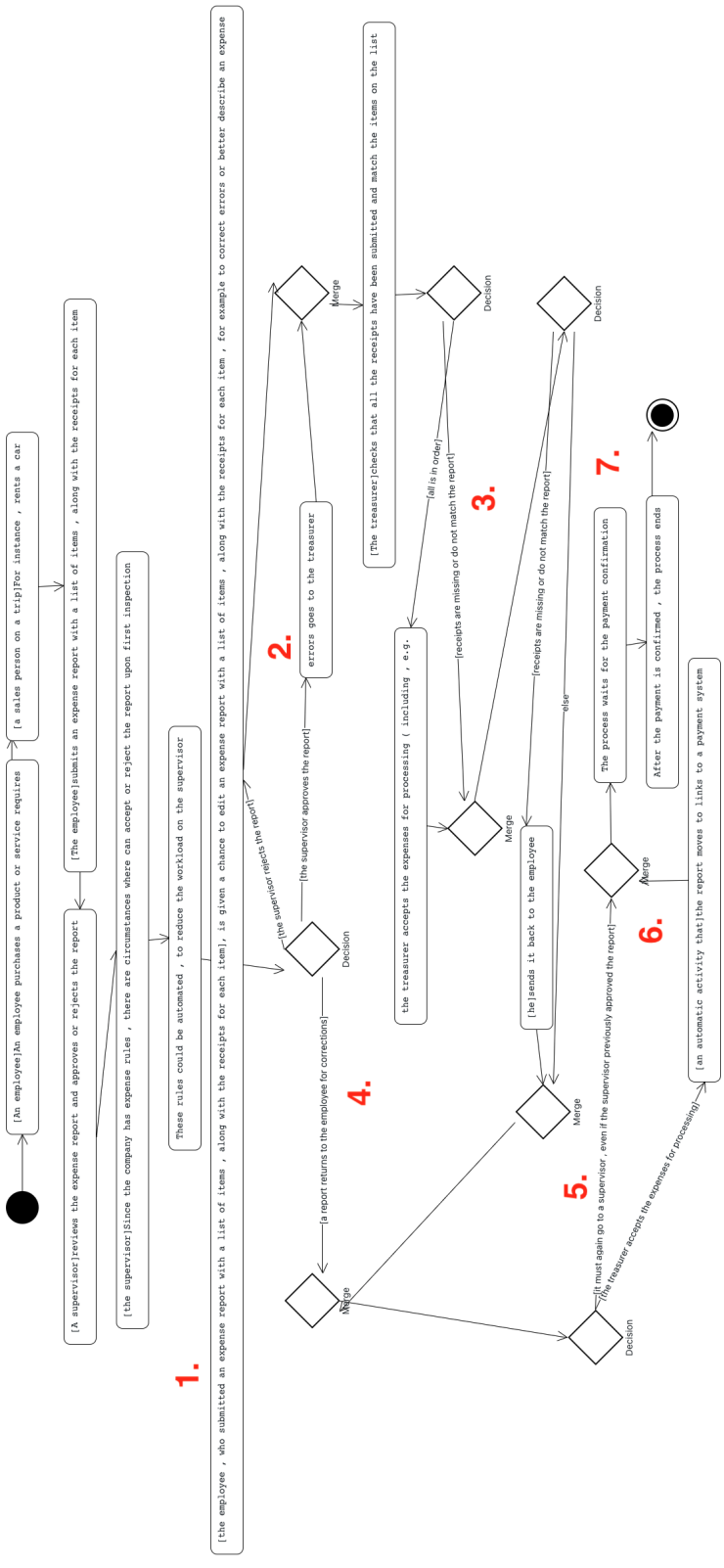

Figure 5.16: *Oracle expense processing example generated by our system with notes.*

The text shown in Example (7) has been taken as input to generate an activity model. In Figure 5.16 we demonstrate the model generated by our pipeline and in Figure 5.15 we show the result from a human modeller that created a model based on the same text. Besides these models, we also have the model from Friedrich et al. which is presented in Figure 5.17. We will first compare our generated model with the input text and the model shown in Figure 5.15.

Figure 5.15 shows the model made by a human modeller. We will look into the shortcomings of our model and compare how it is presented in Figure 5.15. The first Problem 1. has a large name for the actor in this action. This is caused by the replacement of the personal pronoun 'it' in the sentence with its antecedent, which is the following *'expense report with a list of items, along with the receipts for each item'*. In Figure 5.15 the mention of it is not shown or replaced.

The next problem presented by Problem 2., an incorrect coreference replacement of 'it' is presented. The word 'error' should have been the word 'report', again for the human modeller this is not a problem. Then Problem 3. shows a problem with a guard, the guard near the 3.. When we follow the flow further to the next decision node the guard is seen again. So there is a duplicate guard. Formally seen there is nothing wrong with this structure because there is not a final node or a scenario that would hinder the process. Nevertheless, the condition and action following should have been part of the previous decision node.

Problem 4. specifies a correct guard, but the place is incorrect. The guard should not be an alternative path for the conditions presented in the guards about the supervisor's rejection or approval. It should have been a decision node after the action of sending back the report to the employee. The guard has been coupled to these guards because they are also conditions and have the same coreference cluster of the entity 'report'. We do not have a solution currently to solve this, but it could be solved in the future when we identify paths in a text.

Then Problem 5. is a partial effect of problem 4., because the guard in problem 5. should have been an action following the previously mentioned guard. Nevertheless, this would have been created as an action, because our pipeline identified the possible action as a condition. The action was seen as a condition because of the conditional keyword 'if' in the part of the text. It should not have been a condition, as it does not indicate a condition but a saying 'even if'.

Then Problem 6. specifies the two guards that should not be alternative paths to one another. As previously mentioned the condition with the part of text 'even if' should not have been a guard, but an action. Even more,

the two guards should not be alternative paths for one another. This is caused by the conditions that follow one another and the result from our entailment model that specifies that the guards contradict each other. The solution here is to make sure the condition identification works better, to make sure the action was not identified as a condition.

Finally, Problem 7. notes a problem with the order of the actions. The last two actions here should only be executed if the guard *'the treasurer accepts the expenses for processing'* is fulfilled. The alternative path that we have mentioned in the previous problems, enables the flow to also execute the final actions without fulfilling the requirement of the treasurer accepting the expenses. Therefore these actions should be following the action '[an automatic activity that] the report moves to links to a payment system'. As we have seen in the model from Section 5.1.2 there we have the same problem. We are not able to connect some actions to a certain conditional path. That is the same case here.

Another action that we do not see in the text because it is an implicit action is waiting on the response of the employee. This action is shown in Figure 5.15. Although our system does introduce some problems and the actions and actors are not always correctly spelt, it does give somewhat correct actions and keeps the context in those actions.

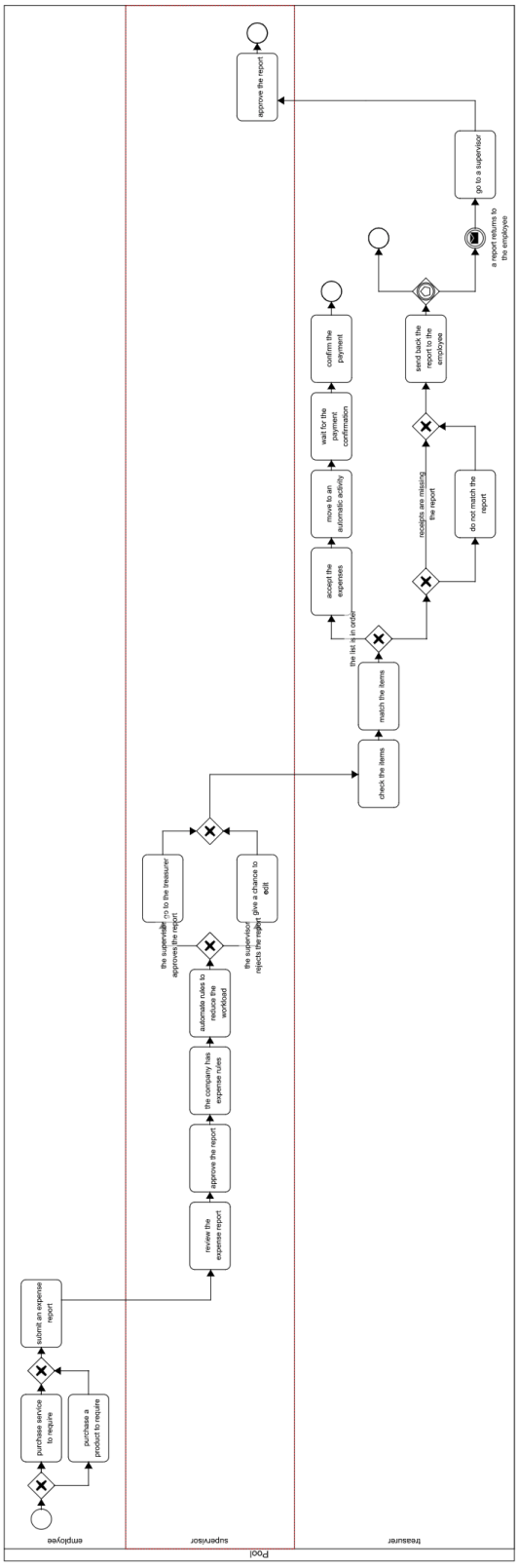

Figure 5.17: *Oracle expense processing example made by a system[24].*

Name	Number of sentences	Source
Ordering materials	13	Business

Table 5.2: *Process description overview from FN-RE.*

5.1.4.1 Difference with generated model

Friedrich et al. also generated a process model for this description, that we present in Figure 5.17. The model of Friedrich is better at identifying the different paths. There are some exceptions with the first alternative path, which should have been two actions following one another. Then the last conditional structure has an incorrect alternative path where there is an action *'do not match the report'* that should have been part of the guard above it. A problem with the order of actions is about the payment process and its' confirmation, these actions should also be executed in the flow where the receipts are missing. So that these actions are executed after the receipts are added and the actions should be executed after the *'approve the report'* action. We see that the model from Friedrich is in some cases better at defining spans of actions following a guard, but it also has its' limitations.

5.2 FrameNet Requirements

The FN-RE dataset is a set of annotated requirements documents labelled using the FrameNet schema[30]. The dataset only contains labelled requirements documents and does not have any process models. Nevertheless, we have selected one interesting process description, that we split into two process descriptions. We generate a model for each process description and compare the generated models with the text itself. For the comparison, we annotate the text with actions, conditions and actions following a condition. We do this ourselves because the annotated frames do not give any additional value for the activity model information.

5.2.1 Ordering materials FN-REQ-015

The ordering materials text specifies the process of ordering materials and the process of preparing a purchase request. We present this text in Example (8).

(8) *The first thing we do is request material using a Purchase Request form. Then the Purchasing department either identifies our current supplier for the kind of material requested or sets out to identify potential suppliers. If we have no current supplier for the needed item, purchasing requests bids from potential suppliers and evaluates their bids to determine the best value. Once a supplier is chosen, Purchasing orders the requested material. Those requesting material must first prepare a Purchase Request. The requester must then obtain the Account Managers approval or that of the designated backup, for the purchase. Purchase Requests submitted for Account Manager approval must include the Account Number for the Project that will fund the purchase. Account Managers or their designated backup, are responsible for, and must approve, all purchases made against their project accounts. After the Account Manager approves the purchase, an authorisation signature may be required. To avoid a potential conflict of interest, the requester cannot be the same individual who approves or authorises the request. Purchase Requests involving Direct projects require an authorisation signature whereas Indirect projects do not. Once all the appropriate signatures are in place, the requester submits the signed Purchase Request to Purchasing. Purchasing then orders the requested material and tracks it as a Purchase Order.*
Ordering materials text[30].

The text can be split into two processes the process of ordering the materials and the process of preparing a purchase request. In Example (9) we describe all the actions and conditions of the order materials process and in Example (10) we do the same for the purchase request. Each sentence in the examples is an action, in he case of a condition or action after a condition, this is indicated by the bold text in front of the sentence.

(9) All actions for the first process on ordering materials.
   a. The first thing we do is request material using a Purchase Request form
   b. the Purchasing department identifies our current supplier for the kind of material requested
   c. the Purchasing department sets out to identify potential suppliers
   d. **Condition**: If we have no current supplier for the needed item
   e. **Action after condition**: purchasing requests bids from potential suppliers and evaluates their bids to determine the best value
   f. **Condition** Once a supplier is chosen

g. **Action after condition**: Purchasing orders the requested material

(10) Actions for the process of approving and preparing a purchase request.

a. Those requesting material must first prepare a Purchase Request
b. The requester must then obtain the Account Managers approval or that of the designated backup, for the purchase.
c. Purchase Requests submitted for Account Manager approval must include the Account Number for the Project that will fund the purchase.
d. Account Managers or their designated backup, are responsible for, and must approve, all purchases made against their project accounts.
e. After the Account Manager approves the purchase
f. an authorisation signature may be required.
g. To avoid a potential conflict of interest, the requester cannot be the same individual who approves or authorises the request.
h. **Condition**: Purchase Requests involving Direct projects require an authorisation signature
i. **Alternative condition**: whereas Indirect projects do not.
j. **Condition**: Once all the appropriate signatures are in place
k. **Action after condition**: the requester submits the signed Purchase Request to Purchasing.
l. Purchasing then orders the requested material and tracks it as a Purchase Order.

We have generated models for each text. First we discuss the model that is generated with Example (9) as input. Figure 5.18 demonstrates the generated model with the problems noted in red. First, Problem 1. is the action *'a supplier is chosen'*, this should be a condition instead of an action. This condition is not found, because there is not a conditional keyword to identify the condition. The keyword we could have used is 'once', but we do not use this word as a conditional keyword. We could add the keyword to our list of conditional keywords to support the search for conditions, but we would need to have more examples to do this.

The next problem is Problem 2., here an action is missing. The action should have been *'Purchasing orders the requested material'*. Why the action is missed is quite unclear, because the part of the sentence does return an

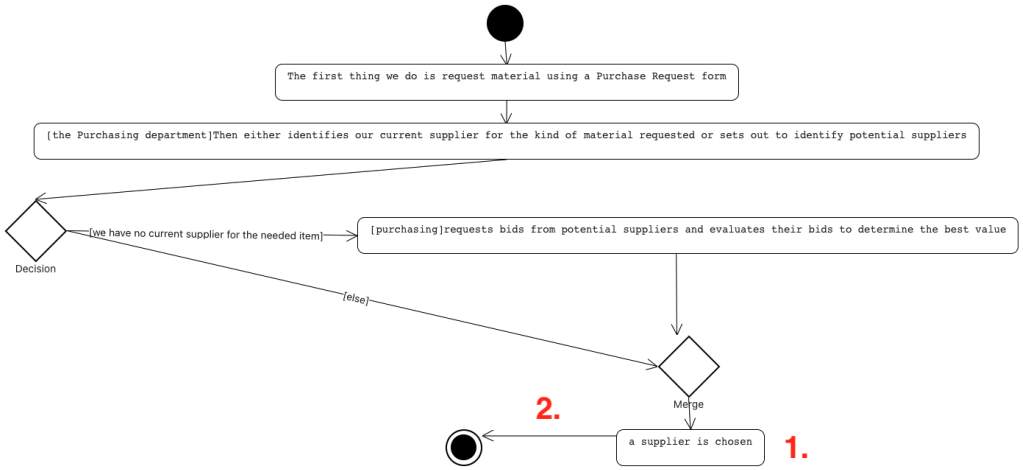

Figure 5.18: *Order materials - order process with notes.*

SRL roleset in the example of the AllenNLP model. We looked into this problem and it shows that the AllenNLP model for SRL that we have used did not return any roleset for the part of the sentence defined in Example (9-g). We have seen such behaviour before with other verbs, such as 'ship'. In that case, we also did not receive an SRL result, while it did when testing it in the demo environment. This might be caused by a bug in the library, we suspect it to be a bug. To solve this we might need to upgrade to a newer version of the AllenNLP library. Aside from these small problems the generated model does mostly model the actions in a correct action and sequence.

The second generated model is shown in Figure 5.19, for this model Example (10) has been used as input. The model is fairly straightforward, with most actions. There are two conditional structures missing at the places of 1. and 3.. These conditional structures are not created, because there is not a conditional keyword that we look for. For the place of 3. the word 'Once' is used again, we have seen this keyword before and it might be a good extension to our set of conditional keywords.

Another problem is shown with 2., here the text of the action is missing. We looked into the roleset of the part of the sentence that was used to generate this action, that is *'whereas Indirect projects do not'*. The SRL tagger returned a roleset for the verb 'do', unfortunately only the word annotated in this frame is 'do'. Thus the action that is created is only 'do'. To solve this we need to enhance our SRL roleset selection approach.

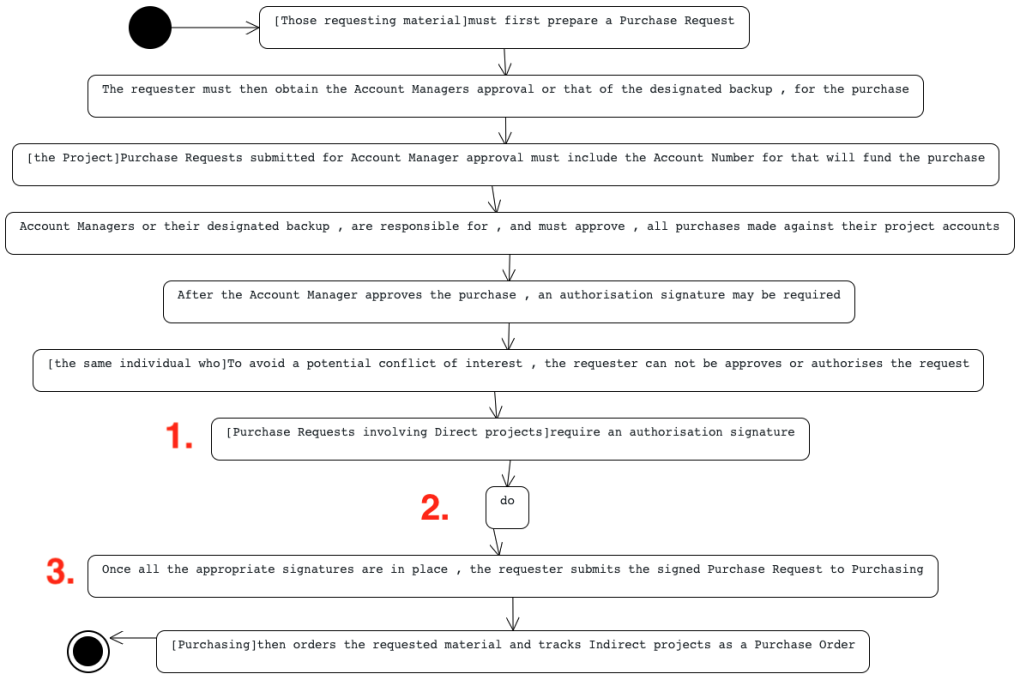

Figure 5.19: *Order materials - purchase request with notes.*

Name	Number of sentences	Source
Validate Temperature	8	Business
Voucher Maintenance System	13	Business

Table 5.3: *Overview of the process descriptions taken from the PURE dataset[31]*

5.3 PURE dataset

The PURE dataset is another dataset with some requirements documents, that we described more in-depth in Section 2.4[31]. The requirements do not have a model that represents the process. Also, we had to manually extract data from the text. Luckily, Martijn Schouten, another student working on another pipeline in the P2P project pointed us to the process texts in the collection of requirements documents. From these results, we have selected two process descriptions, that we will annotate with the actions and conditions. This enables us to compare the generated model with our annotations.

5.3.1 THEMAS - Validate Temperature - SRS-008

The Validate Temperature process description has been taken from a Software Requirements Specification named THEMAS, which stands for The Energy Management System. In Example (11) we present the process description.

(11) *Two types of temperature data shall be recognized from the thermostats: 1) the temperature setting and 2) the current temperature. This module shall process both types of data. A current temperature value that is received from an individual thermostat shall be compared to the valid temperature range values. If the current temperature value is strictly less than the lower value of the valid temperature range or if the received temperature value is strictly greater than the upper value of the valid temperature range, then the THEMAS system shall identify the current temperature value as an invalid temperature and shall output an invalid temperature status. Otherwise, the THEMAS system shall output a valid temperature status. A temperature setting value that is received from an individual thermostat shall be compared to the valid temperature range values. If the temperature setting value is strictly less than the lower value of the valid temperature range or if the temperature setting value is strictly greater than the upper value of the valid temperature range, then the THEMAS system shall identify the temperature setting as an invalid temperature and shall output an invalid temperature status. Otherwise, the THEMAS system shall realize the value for that thermostat's temperature setting.*
Validate temperature description[31].

As we mentioned there is not a model available for this process. Therefore we will manually annotate the text with actions and conditions. This is not a complete annotation, but mostly to identify the main actions and conditions. We present the annotations in Example (12).

(12) Actions for the process of approving and preparing a purchase request.
   a. Two types of temperature data shall be recognized from the thermostats: 1) the temperature setting and 2) the current temperature.
   b. This module shall process both types of data.
   c. A current temperature value that is received from an individual thermostat shall be compared to the valid temperature range values.

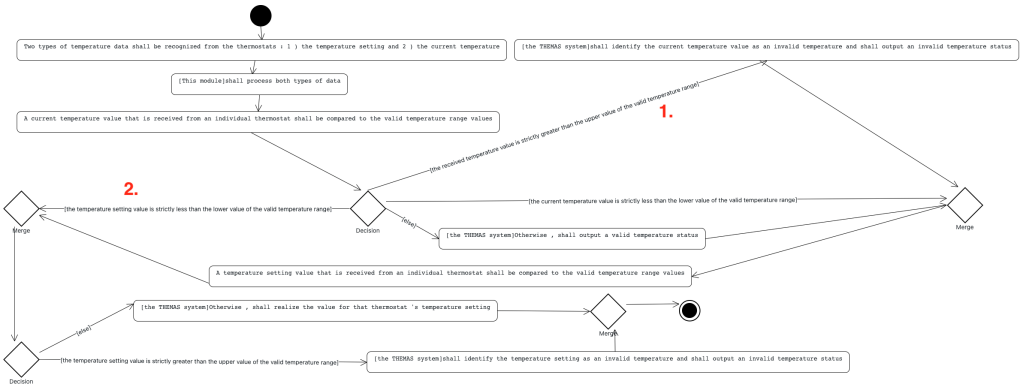

Figure 5.20: *Generated validate temperature model.*

d. **Condition**: If the current temperature value is strictly less than the lower value of the valid temperature range or if the received temperature value is strictly greater than the upper value of the valid temperature range,
e. **Action following condition**: then the THEMAS system shall identify the current temperature value as an invalid temperature and shall output an invalid temperature status.
f. **Alternative action**: Otherwise, the THEMAS system shall output a valid temperature status.
g. A temperature setting value that is received from an individual thermostat shall be compared to the valid temperature range values.
h. **Condition**: If the temperature setting value is strictly less than the lower value of the valid temperature range or if the temperature setting value is strictly greater than the upper value of the valid temperature range,
i. **Action following condition**: then the THEMAS system shall identify the temperature setting as an invalid temperature and shall output an invalid temperature status.
j. **Alternative action**: Otherwise, the THEMAS system shall realize the value for that thermostat's temperature setting.

We will use the annotations from Example (12) to validate our model. The model generated from the input of Example (11) can be found in Figure 5.20. The different red numbers are the problems that we will describe.

Problem 1. specifies a problem with the condition and its transformation into a structure. The condition is stated in Example (12-d), here there are two conditions coupled with the word 'or'. As we have seen in an

earlier result in Section 5.1.3, this type of sentence can specify the guard for two different paths. We split these two conditions to make sure that in the case we should couple them to another earlier specified condition we can couple them. However, in the case of Example (12-d) both conditions should be kept together and both should be a guard to the following action. To solve this problem, the guard next to the 1. should be combined with an 'or' with the guard that states *'the received temperature value is strictly greater than the upper value of the valid temperature range'*.

Then Problem 2. is similar to the previous problem. The condition stated in Example (12-h) is also a conjunction by the word 'or' of two conditions. Just like the previous problem these conditions should have been combined in one guard. Although they look similar, the first structure is different from the previous problem. In this example, the guard is not on the same decision node as with the previous example, because it has a link to the coreference cluster of the previous decision node and is a contradiction to one of the guards in that decision node. The coreference cluster is coming from the entity *'the valid temperature range'* so the coreference is correct, but we do not want to compare these conditions. We do not want to compare them, because they are statements about other entities, namely the *'temperature setting value'* and the *'received temperature value'*. This does indicate that the way we select conditions with coreference clusters to compare using entailment should be extended with a way to identify the important entities in a sentence, such that we only use entailment on conditions that are about the same entity of interest. With this in mind, the contradiction between the guard next to the problem 2. marking is a contradiction with the guard *'the received temperature value is strictly greater than the upper value of the valid temperature range'*. If we look closely at the guards that have been compared for entailment, we see that those are not exact contradictions, because they use other main entities *'temperature setting value'* and *'received temperature value'*. These two entities are very similar, that is why the entailment model has compared these two and noted them as a contradiction. Again the result of the entailment is not 100% correct, but the conditional sentences are very similar. With an extension of the coreference condition comparison, we should be able to also handle the incorrect contradiction, because we only compare sentences that should be compared.

Overall the model and its' actions are correctly created the only problem is multiple conditions that should be modelled in the same conditional structure. Nevertheless, it is a good representation of the given process description.

5.3.2 Microcare - Voucher Maintenance System

Another requirements document from the PURE Dataset is the Microcare document[31]. The document describes the requirements for building a Voucher Management System. The goal of the system is to manage the distribution of vouchers that can be used for the treatment of STDs. VMU stands for the Voucher Management Unit. In Example (13) we present the process description.

(13) *The VMU will create the vouchers and sell it to clients through distributors. The distributor will submit the sales details back to the VMU. Each voucher should have two portions with three tear off voucher slips each for Client and Partner. The client and/or the partner will choose the service provider and will get treatment. First visit is called as Consultation and if the patient is not cured then they can go for first follow up and second follow up. If the patient is not cured then the doctor will refer the patient to some other Hospitals the hospital may be another VSP or any other. Each visit details (including Diagnosis, Lab Test and Drugs) of the patient is called a claim. The VSP will submit the claim to VMIU field office to enter those into the database. The filed office will validate the claim form manually and through system. If any of mandatory information is missed or any false information is existing then the field office will reject the claim back to VSP and the system will keep those claim in a quarantine area. The quarantined forms will be sent back to the VSP for verification, if the VSP returns the claim with satisfactory details, the claims will be entered on to the system, in the following month's batch. Based on the payment terms agreed by VSP, the field office will generate BiMonth or Monthly financial and medical report and send it to MSIU Admin team to arrange the payments for the VSP. To understand the satisfaction of client the MSIU Admin team will get client feedback from some of the clients and send those documents to field office to enter those into database.*
Process description from the Microcare document[31].

As mentioned in the previous text, the PURE dataset does not provide any models with the requirements document. Therefore we will annotate the text with actions and conditions. We will compare the annotated text with the generated model. In Example (14) we present the annotated text.

(14) Actions for the process of approving and preparing a purchase request.
   a. The VMU will create the vouchers and sell it to clients through

distributors.

b. The distributor will submit the sales details back to the VMU.
c. Each voucher should have two portions with three tear off voucher slips each for Client and Partner.
d. The client and/or the partner will choose the service provider and will get treatment.
e. First visit is called as Consultation
f. **Condition**: the patient is not cured
g. **Action after condition**: they can go for first follow up and second follow up.
h. **Condition**: the patient is not cured
i. **Action after condition**: the doctor will refer the patient to some other Hospitals the hospital may be another VSP or any other.
j. Each visit details (including Diagnosis, Lab Test and Drugs) of the patient is called a claim.
k. The VSP will submit the claim to VMIU field office to enter those into the database.
l. The filed office will validate the claim form manually and through system.
m. **Condition**: any of mandatory information is missed or any false information is existing
n. **Action after condition**: the field office will reject the claim back to VSP and the system will keep those claim in a quarantine area.
o. The quarantined forms will be sent back to the VSP for verification
p. **Condition**: the VSP returns the claim with satisfactory details
q. **Action after condition**: the claims will be entered on to the system, in the following month's batch.
r. Based on the payment terms agreed by VSP, the field office will generate BiMonth or Monthly financial and medical report and send it to MSIU Admin team to arrange the payments for the VSP.
s. To understand the satisfaction of client the MSIU Admin team will get client feedback from some of the clients and send those documents to field office to enter those into database.

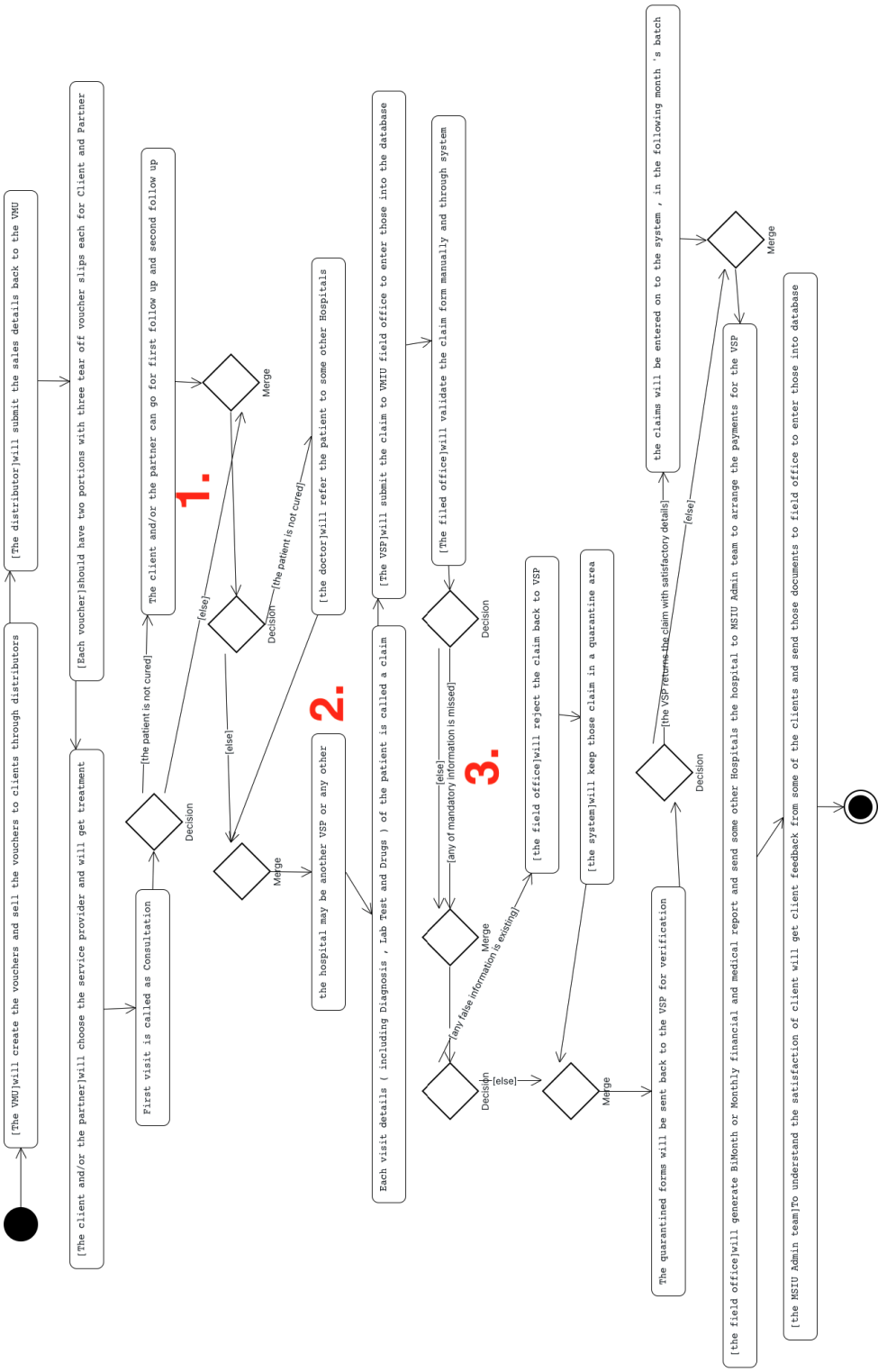

Figure 5.21: *Generated Microcare outline voucher maintenance system model.*

In Figure 5.21 we present the model generated based on the text shown in Example (13). We have added the red annotations to show the problems of the generated model.

The first problem, Problem 1. is an incorrect personal pronoun replacement. The coreference cluster has predicted for the word *'they'* that it refers to the entity *'the client and/or the partner'*, but if we look into the structure and definition of the sentence we see that it does refer to *'patient'*. The incorrect coreference cluster is probably the effect of an incorrect personal pronoun *'they'*. *'they'* is a plural pronoun, while *'the patient'* is a singular pronoun. This difference is what we suspect to be the issue, because when we switch *'they'* with *'he'*, *'she'* or *'he or she'* the coreference finds the cluster of *'patient'*. So it is just a problem with the text.

Problem 2. is the split of the sentence into two actions, that should have been kept together. The division is the effect of a new SRL roleset in the sentence and not in clear conjunction with the word 'and' between the two rolesets. This could be overcome with scanning if a roleset is in the same sentence, but there are cases where we would like to split the two parts.

We have seen Problem 3. before it is a condition that should have been combined with the next condition. The conditions have been split because there are two SRL role sets in the condition statement. The role sets are split with the word 'or'.

To conclude the actions are quite correct and there are problems with some conditional structures. Nevertheless, the problems are solvable and the resulting model is quite correct. Another important aspect here is that the text was not very complex and very structured, because there were not any forward or backward references.

5.4 Industry collaboration

Together with Pegasystems (Pega), we have been working on a collaboration, to see how our work supports their software and the users using it. The goal of the collaboration is to explore the possibilities of our pipeline and try to fit the output into the software of Pega to ease the build processes and applications. In this effort, we have defined together with Pega a text that we can use as a demo scenario and as input for our pipeline. Besides the pipeline, another student Willem-Pieter van Vlokhoven is creating an API mapping software to convert the data generated by our system into a format that can be used in the system of Pega. With this software, we can create an activity model in our system and transform it into a Pega process model. In the following sections, we will demonstrate the text

and the resulting activity models, that we have written and built for this collaboration.

5.4.1 Demo scenario

Pega makes use of the case life cycle design approach. In this approach, several cases are used to specify a process and how it is executed. Within a case, several stages can be executed and within each case there are steps. These steps can be certain actions that are executed. We want to be able to transform our activity model into such a case life cycle. We will not go into the details of how we define this, which will be something the other student Willem-Pieter van Vlokhoven will write more about in the future. We focus on writing the process descriptions and generating activity models for the collaboration.

To follow the case life cycle approach we have defined four different texts. The first text is an overview of the different stages, where each action in the state is a case in itself. In the other three texts, we define the actions that occur in each case.

5.4.1.1 Process overview

The first scenario is an overview of the whole process. We present this text in Example (15). In this text, the different cases are defined and specified in how they interact with one another.

(15) The first stage is the customer captures the order. Then the order manager approves the order. Finally the order is processed by the warehouse manager.

The process description in Example (15) is used to generate an activity model, that we present in Figure 5.22. The model is not very big or extensive, because the input text is brief.

5.4.1.2 Stage models

Based on these different stages we have defined process descriptions for each stage to see what such a process would look like. In Example (16) we present a few sentences for each stage.

(16) Sentences for each stage.

a. A customer enters an order.
Capture order

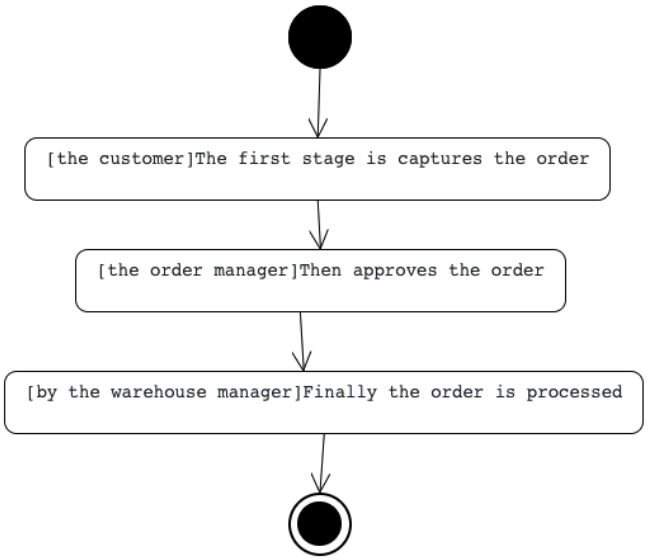

Figure 5.22: *Process overview of order example for Pega.*

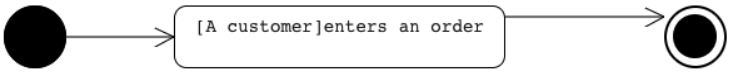

Figure 5.23: *Process of capture order example for Pega.*

b. If the order total is more than 10.000 euros, the order needs to be approved by the manager. If the order is approved, the inventory manager allocates the stock. If the order is not approved, it is cancelled.
Approve order
c. If the order total is less than 10.000 euros, the inventory manager allocates the stock. If the stock level is too low, the product is reordered.
Process order

For each of these stages we have generated a model. The model for Example (16-a) is shown in Figure 5.23, Example (16-b) is presented in Figure 5.24 and Example (16-b) in Figure 5.25.

These stage models are used for the creation of a process model in the Pega system. The models themselves are quite simple, but the goal of them is to try to build a working collaboration. In the future, these can be

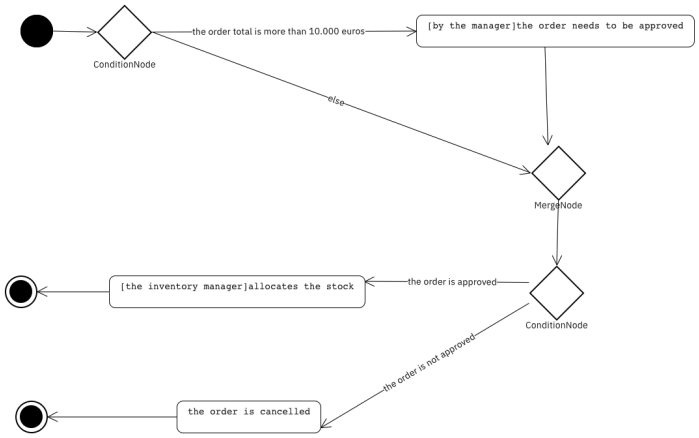

Figure 5.24: *Process of approve order example for Pega.*

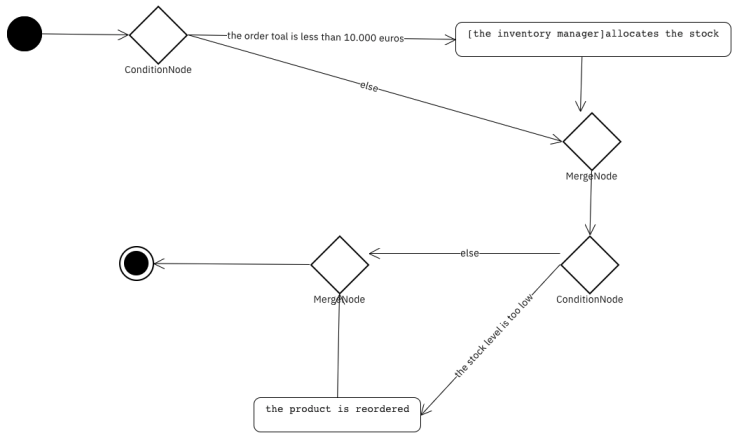

Figure 5.25: *Process of process order example for Pega.*

extended further to be of added value for the creation of processes in low code systems.

5.4.1.3 Complete model

Next to the overview and stage models we also wanted to combine the different stage models, into one larger model. The goal is to identify problems with the smaller models and see how they could be combined. Unfortunately, some parts could not be easily combined, therefore we rewrote some of the parts. Also, we changed some unclear actions that we found while looking into the stage models, such as the stock allocation which only happens when the order is of a certain amount. We could have changed the stage process descriptions also to define them again, but we would like to show that there can be change, such that the stage models can be improved in the future. Therefore we kept the old stage model descriptions which we have shown in Example (16). To show the complete process description we present Example (17).

(17) The process starts with the first activity, where the customer captures order details. This is followed by capture customer details. If the order amount is larger than 10.000 euros, the order needs to be approved by the order manager. If the order is not approved, the order is cancelled. Otherwise, the warehouse manager checks the inventory. If the inventory is too low, the stock is reordered. Finally the warehouse manager ships the order.
Complete demo scenario.

Based on the process description in Example (17) we have generated an activity model. The activity model is shown in Figure 5.26.

These models can be used to generate a process model in the low code platform of Pega. The transformation process will be shown in the future in the work of Willem-Pieter van Vlokhoven. He will create an API mapping layer to transform the models that we have created into Pega models.

The first finding of this collaboration is the difference in ways that vendors such as Pega have built their interpretation of systems, such as the case life cycle. In the case of Pega, they focus on a hierarchical structure, because that is how some of the process descriptions are specified. The process descriptions are explanations of several levels of the process, so there is a high-level description of the stages and a more in-depth description of each stage. The use of stages required us to tailor our activity model to their interpretation of how a system should be defined. This is

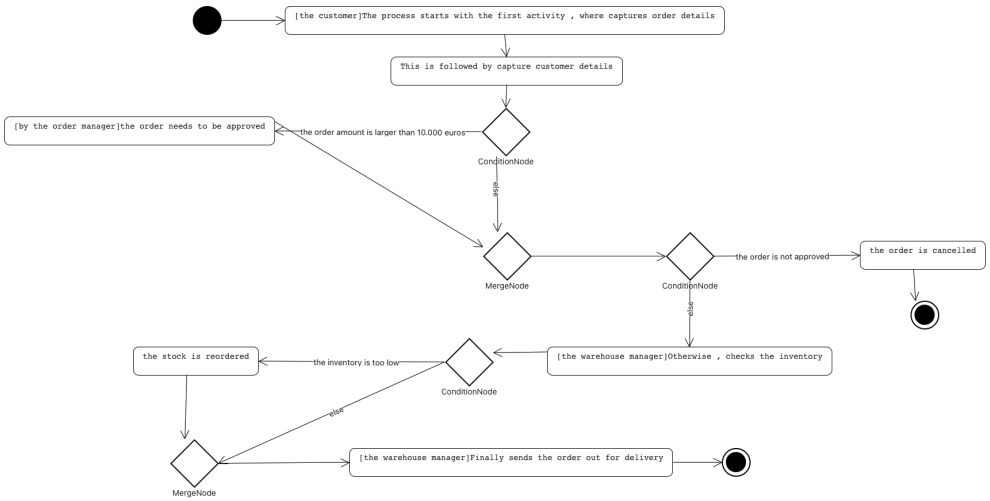

Figure 5.26: *Pega order process*

not problematic, but it requires an extra interpretation step to transform our models into the Pega models. Such a proprietary model and architecture is not only defined at Pega, there are other vendors which made a similar decision or took a whole other approach. Therefore the difference in interpretation of how to define process models and descriptions needs to be taken into account when moving forward in this area.

As we mentioned at the beginning of this section, the collaboration is mostly exploratory. There will be more findings along the way, which will be reported together with other case studies in the future work of Willem-Pieter van Vlokhoven.

5.5 Conclusion

To conclude our results, we have created several models from process descriptions. Some of these models had BMPN models to compare with, and some did not. For the ones that did not have any comparison options, we annotated the text to compare our models.

In comparison with the work of Friedrich et al., we have seen that our pipeline is good at keeping the context of actions and conditions because we do not transform our text. Next to that, the use of our conditional extraction approach with conditional keywords and sentence structures such as adverbials works quite well. The path identification was also good because the coreference and entailment approach enabled us to make more certain decisions about conditions than Friedrich et al. could do. These

choices enabled us to create better conditional structures. Besides these achievements we did see parts where we could improve, such as the condition extraction, actor identification and action identification for conditional paths.

For the other models, we have mostly investigated if the actions and conditions were correctly created. Here we mostly found problems with split-up conditions and the combinations of incorrect conditions that were based on coreference. Also the word 'once' could have been a good extension of the conditional keyword list to find more conditions. Overall the created models represent the processes quite good and there are minor tweaks needed to create a correct activity model.

To summarise the the parts where our pipeline can improve we have defined a list of problems and improvements:

1. Actors
    (a) Incorrect actors;
    (b) Missing actors;
2. Action extraction
    (a) Remove demo or example text;
    (b) Improve rolesets selection, to counter empty rolesets.
3. Condition selection
    (a) Better identification of the adverbials, because they can be split in two.
    (b) Update the list of conditional keywords e.g. add 'once'.
    (c) Condition split in twice with the 'or' word, keep these together.
    (d) Add the ARGM-PNC SRL tag to the condition extraction approach.
4. Path identification
    (a) Improve the condition comparison with coreference and entailment, to search first for neighbouring conditions.
    (b) Improve the identification of actions that should be part of a certain path. e.g. actions part of an alternative path.

The possible improvements are the result of testing our pipeline on a set of unseen process descriptions. Next to these improvements, we would also like to enhance our work with some concepts we have not had the time to touch upon, such as repetition and parallel paths. These are for our future work.

Besides these parts on which we could improve, some parts are quite hard to improve upon. We present these problems as follows:

1. Implicit structures
    (a) actions, which are created using the structure of the model.
    (b) Implicit conditions, for which we need to understand the context of the sentence.
2. References to other processes.
3. Missing SRL rolesets.

With all of these improvements in consideration, our pipeline can be further improved to be better at transforming process descriptions into activity models. Nevertheless, as we have shown the performance of our pipeline does come near the performance of the model of Friedrich et al., still, it is at some points better than our pipeline. Another important aspect of our approach is to note that with the extraction of parts of the text and the creation of activity models we have been working on several tasks all at once. These tasks or problems are path identification, matching and combining conditions, extracting actions and building a correct activity model. What we have seen is that each of these tasks can be a research topic itself. So the results that we have produced are promising and there can be more improvement in this field.

Version of July 9, 2022– Created July 9, 2022 - 15:37

Chapter 6

Discussion

In this thesis, we have presented a prototype that we have created to improve the requirements engineering process concerning the creation of activity models. Next to this prototype, we have demonstrated several approaches to extract information for the creation of these activity models. To be able to do this, we have investigated earlier contributions in this field. We found out that most contributions did not consider activity models, but the more widely used BPMN. Nevertheless, the similarities between these two modelling approaches enabled us to use the contributions from the field of BPMN. From these contributions, we reused some extraction approaches, such as conditional keywords and inspiration for our action extraction. The few available datasets were also helpful in the validation of our approaches.

To improve the existing approaches from the literature we wanted to use state-of-the-art NLP models in our approach. We especially wanted to use the newer transformer models, because they perform well at understanding context. Our first assumption and aim were to use these models as a way to understand our text as a whole and extract activity models through them, but unfortunately, the models were not directly fit for this approach. The limited number of datasets was also a limiting factor for some of the machine learning and natural language processing approaches to construct activity models. Therefore we used transformer models for specific NLP tasks, such as semantic role labelling, coreference resolution and textual entailment. With these tasks, we could extract useful information about the text. The combination of these models and their output enabled us to construct activity models.

The dataset with BPMN models that were available did help us in our validation of our prototype and approaches. In our results we have shown

that our model is good at finding actions and keeping the context, also it is better at combining conditions than the model of Friedrich et al.. Overall the approach for action extraction and combining conditions is a good improvement in the field of process extraction from text. Specifically, the use of coreference and entailment has been a good approach to combining and comparing conditions with more certainty.

To conclude we have presented a prototype to enhance the creation of activity models for the requirements engineering process. These models can help modellers, to quickly create models and easily understand processes, faster than starting with nothing.

6.1 Limitations

There are some limitations to our work, which we will discuss. The first limitation pertains to the structure of texts. As seen in earlier work, there have long been modelling approaches that required some sort of structure in the text. This could range from a text in steps to texts structured in spreadsheet-like formats. In our approach we did not want to work with any structured input, to be able to handle most types of input. Nevertheless, as we have previously mentioned it is hard to understand the sequence of actions within the text. Thus we have assumed that the order of the given text is the order of execution of the process. We do try to overcome this by using coreference for our conditions, to be able to move back and forth when combining process parts in the text. In the future we would like to be able to not be bound to this assumption, but to be able to do this we will need to be able to identify flows of action within a text. At this point, we have not found such approaches.

Another limitation to our approach is that we are not able to process implicit information in a process description. With implicit information we mean information that is clear for a human or expert, but not explicitly stated in the text. Our pipeline relies heavily on all of the information stated in a text and does not have a sense of context, therefore we are not able to handle such information. To be able to work with implicit information our model needs to gather and use such information. This could be a knowledge graph or some sort of domain knowledge input, that the pipeline takes into account.

A point of preference is the way to construct actions. Our approach uses the SRL tags within a sentence to construct the actions that we use. Through this approach, we can extract more information from the sentence, than with only a subject, verb and object extraction. For BPMN

models most activities are presented with brief actions. In activity models, this is not limited to brief actions and longer pieces of text can be used. Because of our transformation, it is often not clear how everything came together, therefore the use of longer actions enables the end-user to understand what an action does and he or she can change it accordingly. Still, this is more of a preference on how the actions should be created.

6.2 Future work

Previously in Section 5.5 we already showed a list of possible improvements for our pipeline. Furthermore, we will propose some possible improvements for the future.

During our research, we found out that the goal of transforming activity models is a combination of several tasks. We would like to propose, similar to Bellan et al. [33], that the research area starts with a separation of the different transformation tasks. Such a task could be action extraction, path identification, condition identification or other tasks. With the definition of these tasks, the problems that we are trying to solve will become more general than only process model specific. Each contribution to one of the tasks will improve the overall performance of the transformation task. Next to that field of research will also be open to researchers that are not experts in the process domain. Still an important thing to take into account about these different tasks, is that they are dependent on one another. We have also seen this in our approaches when we tried to solve one task it sometimes influenced another task.

These different research tasks can be optimised if there is enough data available to validate the approaches with. Therefore we and others[19] with us would like to point out that the research in this area will be improved if there are more datasets available. The datasets can be divided in to several smaller datasets, but there are two main types of output possible for the datasets. The first output type are textual annotations such as output that specify certain nodes or structures for the process models. The other output type is a 'correct' process model.

Each of these outputs should have their own performance metrics. For the annotations this could be a recall and precision, but these metrics are mostly focused on the performance of our model to create the required text. In the process of building nodes there are also textual transformations happening that change the text. The use of recall and precision are then less useful, because the required output for the text is not the same as in the previous text. Therefore the other metrics like similarity might be a

good addition to measure the performance on the textual aspect of the processes.

For the modelling aspects we can make use of the standard precision and recall metrics, but similarity metrics would also be a good addition to compare the generated models with the 'correct' models. These metrics can be quite extensive, but for simplicity the first ones could be distance between nodes, number of nodes and edges and the differences in control nodes such as decision and merge nodes. Another good approach is to use conformance checking to compare the a process model and what is supposed to execute[46].

Overall on the side of datasets and metrics there is a lot to be improved, there have been several approaches to introduce certain metrics[24, 33]. These metrics should be embraced and reused to make sure they will be further implemented and the research can be more easily be compared and quantitatively show where improvements can be made in the field.

Based on the different research tasks, one of the complex things to do is the identification of flows in a text. By flows, we mean the sequence in which the different actions, conditions and other process structures are executed. It is important to identify such a flow, to be able to create the process nodes in the desired order. For conditions, we have proposed a way to combine the different conditions with the use of coreference and entailment, but this is only one type of flow. There are more flows within a text that we would like to identify and process. Therefore the identification of flows is an interesting task that will improve the research area and this work.

On the extension of flows, one approach to identifying and understanding sequence in a text is demonstrative pronouns. Demonstrative pronouns are words like 'this' and 'that'. These words indicate a direction within a text and can be used to understand the flow of a text. The direction of these pronouns can be found using the predictions from a model such as a coreference resolution. Unfortunately, the performance of coreference resolution on the predictions for demonstrative pronouns is quite poor[40]. In the case this performance is improved, the demonstrative pronouns can be used as an indicator of the flow in a text.

As we have previously mentioned our approach for the combination of conditions is based on coreference and entailment. We have seen that it works quite well on the datasets that we have experimented with. However, in the future, we would like to see how these two will perform on more and more complex data. So there is a good opportunity to further research the use of the two NLP models for the identification of condition sets.

An interesting research area that could be beneficial for the generation of process models is the code generation contributions. These contributions introduce models that take natural language text as input to generate code[47, 48]. These models and their techniques can demonstrate improvements for understanding texts and might in the future help the overall generation of process models.

Our approach has been powered by three NLP models and the combination of these models enabled us to build activity models from process descriptions. Another approach for the generation of activity models is to use other types of machine learning models. For example, a model that is trained using activity models and their corresponding process description, such that a new description can be given and a model is generated. Unfortunately, this is quite hard to do, because there is a lot of data needed, but there are not many datasets available. A way that would be more doable is with smaller tasks or subsets of the activity model. Such as conditional structures or parallels. It is easier to train a model for such a task and the results can be combined for the creation of a complete activity model. Still, there needs to be a large amount of data available for this.

To conclude there are many ways to go forward from this point and many areas to further investigate. Each of these to create better activity models from process descriptions!

Chapter 7

Conclusion

Our research presents a new prototype for the transformation of process descriptions into activity models. The transformation approach is a combination of the output of NLP models and rules that define how to transform that output into activity nodes and structures. These NLP models are transformer-based and implemented in the AllenNLP library, so the performance of these models is state-of-the-art or close to that. We have implemented this transformation in an existing environment and made sure that the NLP models can be upgraded and the transformation steps can be extended, to make the solution future-proof. The additional heuristic rules that we use are keyword and structure-based and make use of the outputs of the NLP models, some of which have been reused from earlier contributions. This whole approach enables us to generate activity models.

We have tested and validated our models with the available datasets. In the validation process, we have seen that our model is strong at keeping the context of actions and conditions. Furthermore, the combination of semantic role labelling, coreference and natural language inference enabled us to combine conditions with more certainty than earlier approaches. For future work, we propose that our approach needs to be tested on a larger dataset because there will be more corner cases that need to be solved.

Our solution provides a novel approach to activity model generation from process descriptions and it supports UML modellers in the process of creating activity models.

Bibliography

[1] C. B. Keating and P. F. Katina, *Systems of systems engineering: prospects and challenges for the emerging field*, International Journal of System of Systems Engineering **2**, 234 (2011).

[2] F. P. Brooks, *No Silver Bullet Essence and Accidents of Software Engineering*, IEEE computer **20**, 10 (1987).

[3] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro, *Natural Language Processing (NLP) for Requirements Engineering: A Systematic Mapping Study*, ACM Computing Surveys (CSUR) **54**, 1 (2021).

[4] 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**, 45 (2007).

[5] A. R. Hevner, S. T. March, J. Park, and S. Ram, *Design science in information systems research*, MIS Quarterly: Management Information Systems **28**, 75 (2004).

[6] O. M. Group, *About the Unified Modeling Language Specification Version 2.5.1*, (2017).

[7] G. Booch, J. Rumbaugh, and I. Jacobson, *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*, volume 10, Addison Wesley, first edition edition, 1998.

[8] R. M. Bastos and D. D. A. Ruiz, *Extending UML activity diagram for workflow modeling in production systems*, Proceedings of the 35th Annual Hawaii International Conference on System Sciences , 3786 (2002).

[9] C. V. Geambaşu, *BPMN vs. UML activity diagram for business process modeling*, Proceedings of the 7th International Conference Accounting and Management Information Systems AMIS , 934 (2012).

[10] A. Sahay, D. D. Ruscio, A. Pierantonio, and A. Indamutsa, *Supporting the understanding and comparison of low-code development platforms*, 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) , 171 (2020).

[11] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, (2021).

[12] P. Shi, J. Lin, and D. R. Cheriton, *Simple BERT Models for Relation Extraction and Semantic Role Labeling*, arXiv preprint arXiv:1904.05255 (2019).

[13] J. Ruppenhofer, M. Ellsworth, M. R. L. Petruck, C. R. Johnson, C. F. Baker, and J. Scheffczyk, *FrameNet II: Extended Theory and Practice*, 2016.

[14] M. Palmer, D. Gildea, and P. Kingsbury, *The Proposition Bank: An Annotated Corpus of Semantic Roles*, Computational linguistics **31**, 71 (2005).

[15] A. Williams, N. Nangia, and S. R. Bowman, *A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference*, arXiv preprint arXiv:1704.05426 (2017).

[16] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning, *A large annotated corpus for learning natural language inference*, Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing , 632 (2015).

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need*, Proceedings of the 31st International Conference on Neural Information Processing Systems **2017-December**, 6000 (2017).

[18] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of deep bidirectional transformers for language understanding*, Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) **1**, 4171 (2019).

[19] P. Bellan, M. Dragoni, and C. Ghidini, *Process Extraction from Text: state of the art and challenges for the future*, arXiv preprint arXiv:2110.03754 (2021).

[20] H. van der Aa, C. D. Ciccio, H. Leopold, and H. A. Reijers, *Extracting Declarative Process Models from Natural Language*, International Conference on Advanced Information Systems Engineering , 365 (2019).

[21] X. Han, L. Hu, L. Mei, Y. Dang, S. Agarwal, X. Zhou, and P. Hu, *A-BPS: Automatic Business Process Discovery Service using Ordered Neurons LSTM*, Proceedings - 2020 IEEE 13th International Conference on Web Services, ICWS 2020 , 428 (2020).

[22] R. Sharma, S. Gulia, and K. K. Biswas, *Automated generation of activity and sequence diagrams from natural language requirements*, ENASE 2014 - Proceedings of the 9th International Conference on Evaluation of Novel Approaches to Software Engineering , 69 (2014).

[23] T. Yue, L. C. Briand, and Y. Labiche, *aToucan: An Automated Framework to Derive UML Analysis Models from Use Case Models*, ACM Transactions on Software Engineering and Methodology **24** (2015).

[24] F. Friedrich, J. Mendling, and F. Puhlmann, *Process Model Generation from Natural Language Text*, pages 482–496, Springer, 2011.

[25] K. P. Sawant, S. Roy, S. Sripathi, F. Plesse, and A. S. Sajeev, *Deriving requirements model from textual use cases*, Companion Proceedings of the 36th International Conference on Software Engineering , 235 (2014).

[26] K. Honkisz, K. Kluza, and P. Wiśniewski, *A concept for generating business process models from natural language description*, Knowledge Science, Engineering and Management **11061 LNAI**, 91 (2018).

[27] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, *Feature-rich part-of-speech tagging with a cyclic dependency network*, Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics , 252 (2003).

[28] M.-C. D. Marneffe, B. Maccartney, and C. D. Manning, *Generating typed dependency parses from phrase structure parses.*, Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC) **6**, 449 (2006).

[29] J. C. de A.R. Gonçalves, F. M. Santoro, and F. A. Baião, *Let Me Tell You a Story - On How to Build Process Models*, Journal of Universal Computer Science **17**, 276 (2011).

[30] W. Alhoshan, R. Batista-Navarro, and L. Zhao, *Towards a corpus of requirements documents enriched with semantic frame annotations*, 2018 IEEE 26th International Requirements Engineering Conference (RE) , 428 (2018).

[31] A. Ferrari, G. O. Spagnolo, and S. Gnesi, *PURE: A Dataset of Public Requirements Documents*, Proceedings - 2017 IEEE 25th International Requirements Engineering Conference, RE 2017 (2017).

[32] F. Friedrich, *Automated Generation of Business Process Models from Natural Language Input*, (2010).

[33] P. Bellan, H. V. D. Aa, M. Dragoni, C. Ghidini, S. P. Ponzetto, and F. B. Kessler, *PET: A new Dataset for Process Extraction from Natural Language Text*, arXiv preprint arXiv:2203.04860 (2022).

[34] W. N. Francis and H. Kucera, *Brown Corpus Manual*, Brown University (1979).

[35] L. Quishpi, J. Carmona, and L. Padró, *Extracting Annotations from Textual Descriptions of Processes*, International Conference on Business Process Management , 184 (2020).

[36] K. Lee, L. He, and L. Zettlemoyer, *Higher-order coreference resolution with coarse-to-fine inference*, NAACL HLT 2018 - 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference **2**, 687 (2018).

[37] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy, *SpanBERT: Improving Pre-training by Representing and Predicting Spans*, Transactions of the Association for Computational Linguistics **8**, 64 (2020).

[38] H. Hematialam and W. W. Zadrozny, *Identifying Condition-action Statements in Medical Guidelines: Three Studies using Machine Learning and Domain Adaptation*, (2021).

[39] R. C. B. Ferreira, L. H. Thom, and M. Fantinato, *A Semi-automatic Approach to Identify Business Process Elements in Natural Language Texts*,

Proceedings of the 19th International Conference on Enterprise Information Systems - Volume 1: ICEIS , 250 (2017).

[40] H. Zhang, X. Zhao, and Y. Song, *A Brief Survey and Comparative Study of Recent Development of Pronoun Coreference Resolution in English*, Proceedings of the Fourth Workshop on Computational Models of Reference, Anaphora and Coreference , 1 (2021).

[41] G. J. Ramackers, P. P. Griffioen, M. B. Schouten, and M. R. Chaudron, *From Prose to Prototype: Synthesising Executable UML Models from Natural Language*, 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C) , 380 (2021).

[42] M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. F. Liu, M. Peters, M. Schmitz, and L. Zettlemoyer, *AllenNLP: A Deep Semantic Natural Language Processing Platform*, Proceedings of Workshop for NLP Open Source Software (NLP-OSS) , 1 (2018).

[43] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, and P. G. Allen, *RoBERTa: A Robustly Optimized BERT Pretraining Approach*, arXiv (2019).

[44] R. Driessen, *UML Class Models as First-Class Citizen: Metadata at Design-time and Run-time*, Leiden University. Leiden Institute of Advanced Computer Science (LIACS) , 1 (2020).

[45] T. Tang, *From Natural Language to UML Class Models: An Automated Solution Using NLP to Assist Requirements Analysis*, Leiden University. Leiden Institute of Advanced Computer Science (LIACS) , 1 (2021).

[46] S. Dunzer, M. Stierle, M. Matzner, and S. Baier, *Conformance checking: A state-of-the-art literature review*, Proceedings of the 11th international conference on subject-oriented business process management , 1 (2019).

[47] M. Chen et al., *Evaluating Large Language Models Trained on Code*, ArXiv **abs/2107.03374** (2021).

[48] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, *A Conversational Paradigm for Program Synthesis*, ArXiv **abs/2203.13474** (2022).