

Degree Course

BACHELOR IN INFORMATIK CORSO DI LAUREA IN INFORMATICA BACHELOR IN COMPUTER SCIENCE

From Textual Descriptions to Declarative Constraints: An Application of Large Language Models in Declarative Process Mining

Student

Avezzù Irene

Berichterstatter / Relatore / Supervisor

Donadello Ivan

Zweitbetreuer / Correlatore / Second Supervisor

Maggi Fabrizio Maria

Akademisches Jahr / Anno Accademico / Academic Year

2024/2025



Abstract

Process mining aims to discover, monitor, and improve real processes by extracting knowledge from event logs readily available. It is a practice applicable across diverse fields, including business process analysis.

To effectively comprehend and analyze processes, a fundamental procedure is represented by the discovery phase. During which the declarative constraints are extracted to illustrate the temporal requirements of action execution and are used to define the declarative model. This practice is typically automated through various algorithms that, starting from event logs, recognize patterns, which are subsequently articulated in the form of constraints. However, such data is not always accessible or formatted according to established standards.

Most of the time, experts in the field are available and can provide a comprehensive description expressed in natural language that clarifies the process execution and constraints. Unfortunately, until recent times, there was no possible approach to analyze these textual descriptions and extract the constraints. However, an alternative has emerged with the rapid advancement of generative artificial intelligence, enabling the direct extraction of declarative constraints from textual descriptions of processes.

In this project, we have focused on extracting declarative constraints from textual descriptions of process models utilizing large language models (LLMs). In contrast to prior research in this domain, our work introduces the distinctive element of having tested the LLM with processes derived from a real dataset comprising textual process descriptions. This involved recognizing their strengths, limitations, and potential avenues for future development.

Keywords

Large language models (LLMs)
Generative AI
Process mining
Declarative process models
Constraint discovery
Declare4Py
PET dataset
Business process analysis
Extraction from textual description
Textual process descriptions
Artificial intelligence in process analysis

Table of Contents

1 - Introduction	2
2 - Background	3
2.1 - Basic Concepts of Business Process Management.....	3
2.1.1 - Process Mining	4
2.1.2 - DECLARE.....	5
2.2 - The task of Process Model Information Extraction from Text	6
2.2.1 - Direct Transformation Approach.....	6
2.2.2 - Two-Step Transformation Approach	7
2.3 - Language Models and Large Language Models.....	8
2.3.1 - Transformer-Based Architectures	9
2.3.2 - Prompting Techniques.....	10
2.3.3 - The GPT Family of Large Language Models.....	11
2.4 - The Declare4Py Library.....	12
2.4.1 - Architecture Overview	12
2.4.2 - Supported Functionalities.....	12
2.5 - Natural Language Datasets for Process Model Descriptions	14
3 - Extracting Business Processes with LLMs	15
3.1 - The PET Dataset.....	15
3.2 - Experimental Methodology	16
3.3 - Prompt Design and Description	17
3.4 - Integration in Declare4Py	20
3.5 - Experimental Results	22
4 - Case Studies	23
4.1 - Experiments Set Up	23
4.2 - Case Study with Process 15	24
4.3 - Case Study with Process 35.....	28
4.4 - Case Study with Process 43.....	34
4.5 - Evaluation	39
5 - Conclusion.....	40
5.1 - Observations on LLM Performance and Results	40
5.2 - Further Possible Development	41
Bibliography	42
Appendix.....	44
Prompts.....	45
Case studies: LLM-User Interactions	53

1 - Introduction

“Process mining aims to discover, monitor, and improve real processes by extracting knowledge from event logs readily available” (Van Der Aalst, *Business Process Management Workshops*, 2012). It is a versatile concept applicable across numerous domains. Specifically, within the business context, there exists a necessity to analyze processes with the aim of understanding and optimizing them.

A fundamental element in process mining is a process model that is a formal representation of the process in a standard format, which can belong to two possible expressions: procedural models, where the process is represented as a detailed step-by-step flow, and declarative models, which define constraints that must be respected by the process.

The analysis of processes begins with the discovery phase, wherein event logs, which document all the activities performed to execute the process, are used to extract rules. These serve the purpose of defining a comprehensive model that illustrates all potential execution flows of the process.

This step is commonly automated through various algorithms, including Alpha Miner, Heuristic Miner, and Inductive Miner, among others, which define procedural models such as Petri nets. Additionally, algorithms like MINERful and DecMiner are employed to create declarative models by identifying temporal and behavioral patterns, which are later translated into constraints using the syntax of declarative languages like DECLARE. In this research, we focused on this second type of model.

Several algorithms exist for extracting declarative constraints from event logs, in addition to those previously mentioned. However, one approach that remains underutilized and consequently under-researched is the extraction of declarative constraints directly from textual descriptions of the process provided by subject matter experts. This method is often overlooked due to the extensive time requirement and the necessity for knowledgeable individuals who understand the process, its steps, and its rules; resources that may not always be accessible. For this reason, recently, the research and development of methods to automate this approach have increased.

In contemporary practice, there exists the opportunity to integrate generative Artificial Intelligence into this extraction process. This integration is the primary focus of the current thesis project. A similar project had been developed in the past (Van der Aalst et al., 2024); however, the set of constraint templates has been increased by merging information developed in further research.

To evaluate the skills and performance of Large Language Models (LLMs) in this specific task, we relied on one of the few available datasets that offer annotated descriptions of business processes: the PET Dataset.

In the past, various benchmarks for Process Mining have been developed, such as the one created during the *International Conference on Process Mining*, which focuses on domain knowledge and on different implementation strategies (Berti et al., 2024), but we considered the PET Dataset a more suitable option for our research.

Building on various preceding research, we developed and applied three methodologies for analyzing textual descriptions of processes. This approach provided the LLM with instructions on potential constraints to represent the processes effectively, followed by a thorough analysis of the extracted information.

Recognizing the potential of this technique, we implemented this functionality into one of the most advanced and comprehensive Python libraries for declarative models’ analysis: Declare4Py. This integration provides users with the possibility to analyze textual process descriptions, extracting declarative constraints with the support of an LLM.

The text is structured as follows: The subsequent chapter begins with an introduction to the essential conceptual issues required for a thorough understanding and analysis of the themes that will be explored later. The third chapter presents the fundamental elements of this thesis, including the database utilized for the analysis, the methodology employed, clarifications regarding the development of the prompts used in the analysis, and an explanation of how this task has been integrated into the Declare4Py tool. In the fourth chapter, we conduct a case study on three processes, providing an in-depth examination dataset by qualitatively analyzing the extracted results. Lastly, the fifth chapter includes the conclusions drawn from the analyses conducted and proposes possible future research development.

Additionally, there is an appendix containing the prompts used during the interaction with the LLM as well as the outputs generated from the process analysis conducted in the case study.

2 - Background

This introductory chapter provides the basic notions used in the development of this thesis.

The task of process model extraction from text is a concept that has spread across multiple research fields such as Business Process Management, Natural Language Processing, Machine Learning, and Large Language Models. To better understand the thought process, I provide here the basic notions of these disciplines.

2.1 - Basic Concepts of Business Process Management

Processes are everywhere. Every concept is developed following a series of instructions executed in a concatenative manner. Processes are used by companies to manage the work execution, ensuring a constant performance level.

Business Process Management (BPM) is a discipline that aims to define and optimize business processes. This is achieved through a series of steps that are iteratively executed.

The core element of BPM is a business process, which has been initially defined as “*A collection of ordered activities to model a specific business objective, typically represented graphically using a process model diagram*” (Dumas et al. 2013). Furthermore, a process activity is defined as “*A single unit of work performed in a process; It is an individual step that forms part of the business process*” (Dumas et al. 2013).

Another fundamental element in this field is a *process model*, which has been defined as “*A set of diagrams and supporting information about a process and factors that could influence the process*” (ILBA 2012). The key elements that are usually represented by business analysts in a process model to describe the sequential flow of activities and the current state of a process are:

- *the participants involved in the process*
 - process may be executed internally, within the company, or it may involve several players
 - *the business event that triggers the beginning of the process,*
 - *the activities that happen within the process, and*
 - *the paths and decision points that define the logical flow of the process.*

Process models are typically represented using a graphical representation as *process model diagrams*, offering a quick visualization of the process as a whole to promote optimization. The improvement brings several advantages, such as cost reduction, an increase in quality, an increase in productivity, and standardization, but on the other hand, it requires constant adjustment as the process and the requirements evolve.

“The BPM lifecycle identifies the various phases of a business process and its corresponding information system; process mining potentially plays a role in all phases, except for the implementation phase” (Van Der Aalst, *Process Mining Manifesto*, 2012). The execution flow of all the phases of the Business Processes Management Lifecycle is illustrated in Figure 1.

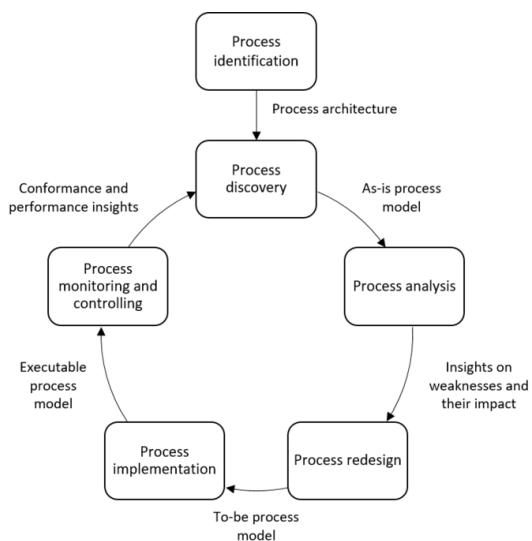


Figure 1: The Business Process Management Lifecycle (Dumas et al., 2013)

The formal representation of the process is the process model, which is developed using a standard format. The discovery phase of the process management lifecycle is responsible for overseeing the identification of the rules of the process and its representation as a model.

When representing a model, there are two possible formats: the *procedural models*, which provide a step-by-step representation of the model using graphical structures such as Petri Nets, and the *declarative models*, which describe the process using temporal rules known as constraints, which identifies precedence, succession and choice relation between the activity of the process.

Out of all the phases required in a business process management lifecycle, this thesis focuses on the first phase: *process discovery*. Unfortunately, as presented by previous research, manual process discovery has several limitations, such as being time and resource-consuming and subject to human error and bias. for these reasons, we analyzed the potential for automating this phase.

2.1.1 - Process Mining

Process Mining is a set of techniques that are applied to event log data in the discovery procedure to extract process information. This is possible by identifying trends, patterns, and details. To reach the goal, this methodology applies specialized algorithms to automatically generate a process model.

An **event log** is a multi-set of *traces*, which are sequences of process activity executed during a process. This structured data, unfortunately, is not always available because oftentimes processes are described in documents in the form of unstructured textual information; this methodology offers easier understanding for non-expert people, but may lead to a more time-consuming discovery process. For this reason, research efforts have been devoted to developing techniques capable of automating the model extraction from text.

A specific element that is commonly used in process mining is a **Petri Net**. Introduced by the German computer scientist C. Petri in 1962, are mathematical and graphical models used to model concurrent systems. These models are in the form of a weighted, directed, bipartite graph in which the nodes represent places and transitions.

A *place* identifies a state of the model and is graphically represented by a circle, which is defined as *marked* if it contains a *token* that represents the actual state of the process instance during run-time.

A *transition* corresponds to a model activity and is graphically represented by a square. They are *enabled* only when all input places are marked, and once they are fired, the tokens are removed from the input place(s) and a new one is generated for the output place of the transition.

One of the first automatic methods to discover a process from event log data was introduced by the **α -algorithm**, which is capable of extracting a workflow net as a Petri-net model that describes the process model encoded in the data. This algorithm is designed to identify the most likely path chosen in a process, find patterns and relations described in the event log, to then use them to build a model of the process.

A process model can be represented using various techniques, such as a **Directly Follows Graph** (DFG), where activities and their ordered relations are represented. Nodes identify the activities while the edges represent the temporal relations between them. The main advantage is the simple and direct translation into a Petri-net or a Business Process Management Notation (BPMN) model without loss of expressivity.

In the past, multiple **formal languages** have been introduced to model business processes, which can be divided into two main categories: imperative and declarative. *Imperative languages*, such as BPMN, represent all the possible paths from start to end. On the other hand, *declarative languages* summarize the relations between activities in terms of constraints. Declarative languages are based on the presupposition that what is not constrained is permitted. We briefly introduce them in the following Sections.

2.1.2 - DECLARE

Traditional process modeling languages based on an imperative paradigm, utilizing a "closed world", facilitate the understanding for stakeholders but are unable to support non-regular processes where different ways of executing a process are possible. As a solution to this, declarative process modeling languages ground themselves on the assumption of an "open world" where everything is allowed unless it is explicitly prohibited.

DECLARE (Petic et al., 2007) is recognized as one of the most established declarative process modeling languages. Its most well-known characteristic is the representation of the process with the support of temporal constraints among activities, which specify which activities can or cannot be performed. These *temporal constraints* are grounded on Linear Temporal Logic over finite traces (LTL_f).

The **DECLARE constraints** are constructed starting from a template. Table 1 presents some of the most common ones.

Family: Template	LTL _f Semantics	Description
\mathcal{E} : EXISTENCE (n, A)	$F(A \wedge X_{\text{existence}}(n-1, A))$	A has to occur at least n times.
\mathcal{E} : ABSENCE (n + 1, A)	$\neg_{\text{existence}}(n+1, A)$	A has to occur at most n times.
\mathcal{E} : EXACTLY (n, A)	$\text{existence}(n, A) \wedge \text{absence}(n+1, A)$	A has to occur exactly n times.
\mathcal{E} : INIT (A)	A	Each case has to start with A.
\mathcal{C} : CHOICE (A, B)	$FA \vee FB$	A or B have to occur at least once.
\mathcal{C} : EXCLUSIVE CHOICE (A, B)	$(FA \wedge \neg FB) \vee (\neg FA \wedge FB)$	A or B have to occur at least once but not both.
\mathcal{PR} : RESPONDED EXISTENCE (A, B)	$FA \rightarrow FB$	If A occurs, B must occur as well.
\mathcal{PR} : RESPONSE (A, B)	$G(A \rightarrow FB)$	If A occurs, B must eventually follow.
\mathcal{PR} : ALTERNATE RESPONSE (A, B)	$G(A \rightarrow X(\neg A \cup B))$	If A occurs, B must eventually follow without any other A in between.
\mathcal{PR} : CHAIN RESPONSE (A, B)	$G(A \rightarrow XB)$	If A occurs, B must occur next.
\mathcal{PR} : PRECEDENCE (A, B)	$(\neg BUA) \vee G(\neg B)$	B can occur only if A has occurred before.
\mathcal{PR} : ALTERNATE PRECEDENCE (A, B)	$(\neg BUA) \wedge G(B \rightarrow X((\neg BUA) \vee G(\neg B)))$	B can occur only if A has occurred before, without any other B in between.
\mathcal{PR} : CHAIN PRECEDENCE (A, B)	$G(X(B) \rightarrow A)$	B can occur only immediately after A.
\mathcal{NR} : NOT RESPONDED EXISTENCE (A, B)	$FA \rightarrow \neg FB$	If A occurs, B cannot occur.
\mathcal{NR} : NOT RESPONSE (A, B)	$G(A \rightarrow \neg(FB))$	If A occurs, B cannot eventually follow.
\mathcal{NR} : NOT PRECEDENCE (A, B)	$G(FB \rightarrow \neg A)$	A cannot occur before B.
\mathcal{NR} : NOT CHAIN RESPONSE (A, B)	$G(A \rightarrow X(\neg B))$	If A occurs B cannot occur next.
\mathcal{NR} : NOT CHAIN PRECEDENCE (A, B)	$G(XB \rightarrow \neg A)$	A cannot occur immediately before B.

Table 1: DECLARE constraints in LTL_f

The use of DECLARE constraints allows modelers to avoid directly formalizing the model in LTL formulae. Indeed, once the templates have been instantiated, the semantics are automatically specified.

Since DECLARE is not a fixed language, it allows the creation of specific languages referring to their own domain.

There are generally only two main **categories of constraints**: unary and binary constraints, but there is also the chance to create custom constraints involving multiple activities.

A *unary constraint* involves a single process activity, e.g., the *existence (A)* constraint reported in Table 1.

On the other hand, *binary constraints* define a connection between two process activities known as *activation* and *target*. The presence of the activation activity activates the verification process of the constraint.

When a DECLARE model is deployed, errors may happen during the run-time of the model; for this reason, they are checked against three types of errors:

- *Presence of dead activities, which are those activities that will never be performed,*
- *Presence of conflicting constraints, where the execution of one will violate the other constraints,*
- *History-based violations may occur if the latest version of the model conflicts with the original version.*

All DECLARE constraints can be grouped into two types: *mandatory* and *optional*, based on whether they may or may not be violated during the execution.

2.2 - The task of Process Model Information Extraction from Text

The extraction of a process model from texts presents a multifaceted challenge since it must account for various linguistic levels, including syntax, semantics, and pragmatics, while also addressing linguistic phenomena such as syntactic variability. Furthermore, it must consider the numerous potential interpretations, manifested as distinct process models, that may be drawn from a single text.

Process model extraction from text is defined as the process of finding an algorithmic function to generate a process model diagram from its procedural description in natural language.

When analyzing a process description, the first step is to *filter out uninformative sentences*, as not every sentence pertains to the elements of the process. After that, we can *extract key elements* such as Actions, Actors, Events, Gateways, Artifacts, and different process flow objects. It's important to keep in mind that each sentence can narrate multiple process elements, and individual words may have different meanings.

To effectively determine the intended meaning and connect it to the correct process element, it is important to take into consideration these nuances. Additionally, once the process elements have been identified, it is important to organize them logically based on the semantics of the description, establishing the *logical succession of process model elements*.

The task of *process model extraction from text* involves converting a procedural description into a visual process model diagram using an algorithmic function, denoted as f .

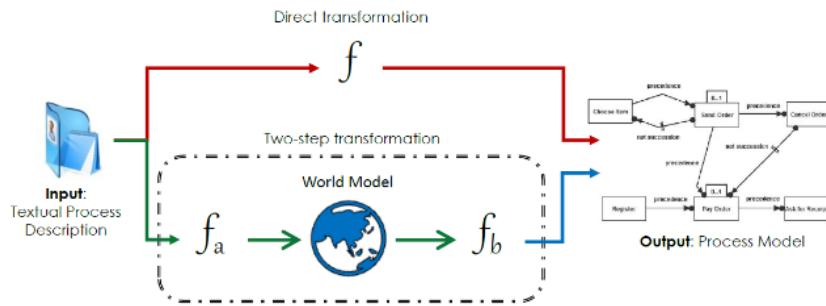


Figure 2: Process model extraction from text

As illustrated in Figure 2 (Bellan et al., 2021), there are two primary methods to tackle this challenge.

The first approach focuses on a direct transformation from the input description to its corresponding process, such as a DECLARE process model. This methodology has been selected for application in the development of the analysis later presented in this research.

The second method takes a *divide-and-conquer* approach by splitting f into f_a and f_b using an intermediate representation known as the *world model*.

Both approaches are further presented in the following sections.

2.2.1 - Direct Transformation Approach

Solutions in this first category focus on *directly mapping* a process description into its corresponding process model representation using a single function, f . This concept is visually represented in the upper section of Figure 2. The function f is often performed through a *complex and ad-hoc tailored pipeline*. This method brings the benefit of a specific transformation process that can leverage all relevant contextual information to address the issue at hand. However, this advantage can turn into a limitation when it comes to developing general solutions or applying the algorithmic function f in varied contexts. Another method for achieving a direct mapping with function f is the use of artificial neural networks. Yet, due to the extensive data requirements for training a transformation model, paired with the limited data often found in this research area, this approach is rarely utilized.

2.2.2 - Two-Step Transformation Approach

The second method previously discussed involves a *two-step transformation* to extract and construct a process model, utilizing a *world model* as an intermediary representation.

As shown in the lower section of Figure 2, we can think of the algorithm as a *compound function* f , which is expressed as $f_a \circ f_b$. In the first step, the function f_a extracts process elements from the text and populates the intermediate representation. In the next step, function f_b takes this structured representation to create the process model diagram based on the elements derived from the world model.

It's important to recognize that both the f_a and f_b components can be subdivided into smaller, more manageable tasks to better address the complexity of the problem at hand.

2.3 - Language Models and Large Language Models

A **language model** is a probabilistic model of a natural language, i.e., English, representing a probability distribution of words, word sequences, or a specific word sequence. They are typically used in speech recognition and information retrieval.

There are different possible strategies to *develop a language model*.

Word n-grams language models are based on the concept that the likelihood of the next word in a sequence relies solely on a fixed number of preceding words. However, this approach faces a challenge known as the *zero-probability* problem, where words that are not present in the model are assigned a probability of zero. Because the overall probability of a sentence is calculated by multiplying the probabilities of its words, any sentence containing an unseen word ends up with a probability of zero.

On the other hand, the **skip-gram models** address the issue of data sparsity by using embedding vectors to represent words. In this framework, the semantic relationships between words are captured through their linear combinations. Unlike n-gram models, skip-gram models show a degree of compositionality, providing a richer representation of language.

Each word is represented by a vector, and each unfamiliar word can be derived by the linear combination of vectors belonging to well-known words. An example could be the vector of the word symphony defined from the combination of the vectors of the words novel, author, and composer.

$$\vec{v}_{\text{novel}} - \vec{v}_{\text{author}} + \vec{v}_{\text{composer}} \rightarrow \vec{v}_{\text{symphony}}$$

However, it's important to acknowledge that this modeling approach faces the *curse of dimensionality*, which is the consequence caused by the number of possible sequences of words increasing exponentially with the size of the vocabulary, causing a data sparsity problem (Bellan, 2024).

Continuous space language models addressed this issue by representing word embeddings as non-linear combinations of weights within a neural network framework, particularly through Recurrent Neural Networks (RNNs), like Long Short-Term Memory (LSTM) networks. Still, RNNs can be quite complex to train and often demand large datasets, which may not always be accessible.

In comparison, **Transformer-based Large Language Models** (LLMs) are a collection of language models built upon artificial neural networks that utilize a transformer architecture. This architecture harnesses attention mechanisms to determine the relevance of a word to other words within the context windows of its neighborhood words.

The LLMs leverage a *transformer-based methodology*, utilizing massive artificial neural networks trained on substantial amounts of textual data, which are, in the vast majority, unlabeled. They excel in various tasks, ranging from language comprehension to text generation. By taking inputs written using natural language, referred to as prompts, these models can produce text that effectively addresses specific problems. They are skilled at capturing the nuances of syntax and semantics, as well as the inaccuracies and biases contained within the training data. Prominent examples of transformer-based LLMs include GPT-3, GPT-4, PaLM, and LLaMA, all of which have demonstrated human-like performance in various applications.

The GPT models are available for user usage through the web app developed by OpenAI, known as ChatGPT, at the following link <https://chatgpt.com/>.

Furthermore, the PaLM, developed by Google, finds an interesting application in Minerva, the first Italian-centric AI language model, i.e., pretrained on only Italian texts. More information regarding this project, developed by Sapienza Università di Roma, is available online at <https://nlp.uniroma1.it/minerva/>.

The LLaMA model is introduced in section 2.3.3 *The GPT Family of Large Language Models*, where a more in-depth refiguration of this model family is presented.

When it comes to training these language models, two primary strategies are employed:

- *Masked Language Modeling (MLM): this approach involves randomly masking a portion of the input tokens in a sentence and training the model to predict these masked tokens.*
- *Causal Language Modeling (CLM): also known as autoregressive language modeling, this method trains the model to anticipate the next tokens in a sequence based on previous tokens and is particularly valuable for text generation tasks.*

2.3.1 - Transformer-Based Architectures

A **transformer** is a type of neural network architecture that utilizes attention mechanisms to create hidden representations of tokens by assessing the significance of each token in relation to the current one. This approach has demonstrated remarkable efficiency across various domains, including natural language processing (NLP) and computer vision. At the core of this architecture are *scaled dot-product attention* units. This mechanism emphasizes relevant tokens while downplaying irrelevant ones (Vaswani et al., 2017). The scaled dot-product attention accepts three matrices as input:

- *Q matrix*: the query vector, i.e., the word we are analyzing
- *K matrix*: the key vector, i.e., the most relevant word related to *q*
- *V matrix*: the value vector

To understand a word *q* within a sentence, the attention mechanism finds the most relevant word *k* related to *q* through a probability distribution.

The output from the attention unit is a matrix of values *V*, which represents the learned vector derived from the input calculations; it offers insights that may further elucidate the relationship. The **attention mechanism** is composed of three evaluations:

1. *Dot Product*, also known as matrix multiplication, between queries *Q* and keys *K*
2. *Scaling function* to scale the values
3. *Softmax function*, which provides a probability distribution of weights over the values *V* when applied to the dot product

The **softmax function** generates output probabilities that, when multiplied by the *V* vectors, produce the **attention value** corresponding to a specific key. Subsequently, another matrix multiplication takes place between the value matrix *V* and the calculated attention matrix.

This evaluation can be expressed with the following formula

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{d_k}V\right)$$

Which can be visually represented as shown in Figure 3.

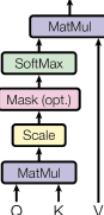


Figure 3: Visual representation Attention value evaluation

These components are then combined to compute the attention as:

In the *Generative Pre-Trained Transformer* (GPT) models, the *Q*, *K*, and *V* components are derived from the same data sources, a process referred to as *self-attention*, which evaluates the relevance of each element's matrix in relation to all the other elements belonging to the same sentence or text. This approach allows the consideration of the context as a whole, improving the understanding capabilities of the model.

A **transformer-based model** generally comprises two main components:

Encoder: is a neural network that iteratively creates a more abstract and semantically rich contextual representation for each input token

Decoder: It utilizes a self-attention mechanism along with an attention mechanism (sometimes labeled as encoder-decoder attention) to retrieve relevant information from the encodings produced by the encoders.

Unlike supervised models, the decoder transformer operates as an autoregressive structure, meaning it doesn't incorporate any future information about a token. The last layer consists of a linear transformation followed by a softmax layer, which predicts the probabilities of the most likely token for each token position. This method enables the language model to generate coherent text.

There are three primary types of ***transformer-based architecture***:

- ***Encoder only:*** it involves a full encoder and full decoder, with the BERT model (Bidirectional Encoder Representations from Transformers, by Google, 2018) exemplifying this architecture
- ***Encoder-decoder:*** It consists of a full encoder paired with an autoregressive decoder. Here, the encoder gathers the context of the input sequence and forwards it to the decoder, which produces the final output sequence. The T5 family (by Google AI, 2019) is representative of this architecture.
- ***Decoder only:*** it features an autoregressive encoder alongside an autoregressive decoder, which characterizes the GPT family (by OpenAI, 2018).

2.3.2 - Prompting Techniques

A ***prompt*** serves as a structured input in natural language for an LLM, guiding it on the tasks it needs to accomplish with a specific text. Typically, a prompt includes clear task instructions, details on the desired output format, and specifications regarding tone and writing style. For example, it might direct the model to produce results as a bullet point list or as an elaborate description. The act of creating these prompts is known as ***prompting***, which involves organizing text effectively to ensure the model can execute the task successfully.

Prompts might also feature examples that illustrate the task to be addressed. This method falls under a concept called ***in-context learning***. In-context learning refers to the capability of a model to understand and execute a task by conditioning its responses based on provided examples within the prompt. This ability emerges naturally from the model and resembles few-shot learning; however, unlike few-shot learning, it does not modify the model's internal parameters.

Various ***techniques for prompt engineering*** have been introduced in academic discourse, including Chain-of-thought, Generated knowledge prompting, Self-refine prompting, Tree-of-thought prompting, and Automatic prompt generation.

Chain-of-thought (CoT): is a technique involving guiding the model to tackle a problem by outlining intermediate reasoning steps necessary for arriving at a solution. This approach emulates human thought processes during reasoning. The primary benefit of CoT is that it compels the model to consider each step of reasoning, resulting in enhanced performance on tasks that require logical thinking, complex arithmetic, and commonsense reasoning. CoT has shown effective results even in *zero-shot prompting* settings, i.e., without in-context learning, by starting prompts with "Let's think *step-by-step*."

Generated knowledge prompting: this technique encourages the model to produce relevant facts that can complete a given prompt. The information generated is then utilized to finalize a prompt, which is presented to the model to extract the desired knowledge.

Self-refine prompting: this strategy requires the model to evaluate its own solutions and use those critiques to refine a final prompt that will serve as the input for deriving the resolution to a problem.

Tree-of-thought (ToT): this technique generalizes CoT by prompting the model to create multiple potential reasoning steps rather than just one. Which are then used to formulate answers, with the final solution being derived through a tree search algorithm applied to the answers.

Automatic prompt generation: this technique utilizes either two separate LLMs or the same model twice to create efficient prompts automatically.

A ***prompting LLM*** is utilized to produce the task instructions. The user provides examples of input-output pairs for this LLM, which analyzes these pairs to infer the potential task instructions that could have led to the generated output. The instructions, crafted by the LLM, are then converted into a new prompt, which is fed into a target LLM, that provides the solution to the problem.

2.3.3 - The GPT Family of Large Language Models

In 2018, OpenAI introduced the first transformer-based model, the **Generative Pre-trained Transformer** (GPT-1) (Radford et al., 2018). This model featured a twelve-layer decoder architecture with 117 million parameters, showcasing the effectiveness of self-attention mechanisms. GPT-1 excelled in capturing contextual relationships in text and exhibited impressive language generation abilities.

After its first model, OpenAI released a second, third, and also a fourth version, amplifying every time the volume of data used in the training process of the models, consequently improving the performance of the model. The abilities of the OpenAI models and their quick success in various fields have led to an increase in attention regarding LLMs.

To keep up with the innovation in 2023, Meta AI released its first version of **Llama**, aiming to support writing processes. Llama, which stands for *Large Language Model Meta AI*, is a family of large language models that has now reached its 4 model.

Before it was first released, Meta AI suffered from a *leak* of the model when unauthorized copies of the first model were shared via BitTorrent. Later versions of Llama became available beyond academic institutions and were launched under licenses that allowed for *limited commercial use*.

In this thesis, I have utilized two of the latest models of Llama: Llama-3.1 and Llama4-maverick.

Released in the summer of 2024, **Llama-3.1** represents the updated version of Llama-3, released just a few months before it. This model was developed as a standard decoder-only transformer architecture, using *405 billion parameters*. It is capable of supporting a *128K token context window* and shows great performance in tasks such as code generation, reasoning, and long-form text synthesis.

The latest version of Llama is represented by **Llama-4**, which, since April 2025, has been offering two relevant versions of the model: Llama4-scout and Llama4-maverick. Both versions have been developed to handle multimodal tasks and employ a mixture-of-experts architecture; they differ in scale, specialization, and intended use cases. Furthermore, both models are multilingual and capable of handling prompts in 12 different languages.

Multimodal tasks belong to a deep learning technique in which the input is provided with multiple types of data, such as text, audio, images, or video. Multimodal models are able to handle all the different formats of input and combine them to achieve the best performance. In this case, Llama4 supports text and image input but only produces text output.

Mixture-of-experts (MoE) is a machine learning approach that separates the model into different subnetworks (called experts), each specializing in a subset of the input data, to perform a task together.

Llama-4-Maverick uses 400 billion parameters made up of *17 billion parameters* per input, combined with *128 experts*, which are able to handle a *context window up to 1 million tokens*, versus the 10 million tokens of Llama-4-scout. **Llama-4-Scout**, smaller compared to its successor Maverick, this model uses a total of 109 billion parameters formed by *17 billion active parameters* and 16 experts, managing up to 10 million context length. These models appear to be particularly suited for complex, general-purpose AI applications requiring robust performance across diverse domains.

2.4 - The Declare4Py Library

A key element in process mining is a process model that is a formal representation of the process in a standard format. This can be done with two possible modalities: procedural models and declarative models.

Procedural models specify a detailed step-by-step flow, making them less suitable for processes that involve numerous branches and exceptions.

Declarative models offer a more direct approach because they simply outline a series of constraints that the process must adhere to.

Available on the market are multiple process mining tools, both commercial and academic, to handle the development and analysis of procedural models. On the other hand, for declarative models, only a few (Java-based) libraries are available.

The concept of implementing Declare4Py (Donadello et al. 2022) arises from this identified gap in the market.

2.4.1 - Architecture Overview

Initially published in 2022, Declare4Py is an innovative and simple-to-use Python library that provides multiple APIs to support the development and analysis process of Declarative models.

The code has been developed to address the core declarative process mining tasks based on Declare [Chapter 2.1.2], utilizing the MP-Declare language extension. This extension accommodates both data- and time-aware constraints, as well as control-flow constraints. As a pioneer in its field, this offers support in the phases of conformance check, process discovery, and query control.

Developed in Python, Declare4Py easily integrates with leading Machine Learning frameworks such as SKlearn, Tensorflow, and Pytorch. Moreover, it's being a library allows for direct invocation, making it ideal for handling extensive experiments.

The latest version of this library can be found online at <https://github.com/ivanDonadello/Declare4Py/tree/main>. In addition to the library code, the authors have made several Jupyter notebooks available to further clarify the key elements, classes, and methods of Declare4Py.

2.4.2 - Supported Functionalities

To handle all its functionalities, Declare4Py relies on some specific **standards** for **input** and **output** files, such as XES for event logs and decl for the Declare models, ensuring compatibility with other libraries.

Before presenting all the tasks performed by this library, it is important to recall some previously defined concepts.

A *trace* σ is a completion of a business process; it is made of a series of *events*, each of which relates to the fulfillment of an *activity* a executed at a time t . Each activity can be characterized by a specific attribute (or set of attributes) known as event payloads. A *log* L is a set of multiple traces.

A Declarative model $\mathcal{M} = \{\varphi_1, \varphi_2, \dots\}$ is defined by a set of Declare constraints created following parameterized templates defined as \mathcal{A} .

A trace satisfies a model $\sigma \models \mathcal{M}$ if and only if each constraint of the model is satisfied $\varphi \in \mathcal{M}$, i.e., $\forall \varphi \in \mathcal{M}, \sigma \models \varphi$.

As previously mentioned, the Declare4Py library supports *three main tasks* of model development: conformance checking, process discovery, and query checking.

The **Conformance Checking** task is appointed to verify, given a log L of traces σ_i , if they satisfy an MP-Declare model \mathcal{M} by testing each trace $\sigma_i \in L$ for all constraints of the model.

To perform this control, Declare4Py uses the *approach* proposed by MP-Declare, receiving the model and log as input and returning the number of activations, fulfillments, and violations for each one of the constraints from the model, repeating the steps for every single trace of the log.

The *results* are organized in a Python data structure indexed by trace identifier, making it easily accessible to the user for information retrieval and aggregation.

The next supported task is **Process Discovery**, which aims, from a log L of traces σ_i and a support threshold ths , to compute and return a Declare8 model \mathcal{M} whose constraints satisfy a percentage of traces in L higher than or equal to ths . Formally, this task can be defined as:

$$\mathcal{M} = \{\varphi : |\{\sigma \in L : \sigma \models \varphi\}| / |L| \geq ths\}$$

Declare4Py implements this task using a *two-step approach*: initially extracts all activities (and pairs of frequently occurring in L with the support of the Apriori algorithm to later use in the development of the model. In the development phase, all possible constraints \mathcal{C} are generated following the provided constraints templates, and then a subset of them is used to generate the model ($\mathcal{M} \subseteq \mathcal{C}$).

Similarly to the previous task, the *results* are provided using a data structure, exportable as a `decl` file, containing for each constraint in \mathcal{M} the traces that satisfy it.

Lastly, the **Query Checking** tasks support the automatic discovery of meaningful temporal constraints that frequently occur in the log.

This task takes a log L , a support threshold ths , and a query q , which is a constraint template where the activation and/or the target activity are unspecified.

Given a set of all possible activities $Vars$, these tasks test the validity of q for every possible assignment ($\lambda : Vars \rightarrow A$), returning the assignment that makes the query meaningful ($\Lambda = \{\lambda_1, \lambda_2, \dots\}$). Only the constraints that have several assignments that are greater than or equal to the threshold are returned. This task can be formally defined as:

$$\Lambda = \{\lambda_i : |\{\sigma \in L : \sigma \models q[\lambda_i]\}| / |L| \geq ths\}$$

The *results* are a data structure containing the assignments.

2.5 - Natural Language Datasets for Process Model Descriptions

Before the creation of the PET Dataset, multiple experts proposed various solutions to resolve the absence of an annotated gold standard dataset. However, each one of them presented big limitations regarding the volume of data, the accuracy of annotations, and the scope of the proposed processes. The following are some of the most relevant.

Friedrich et al. (2011) proposed the first dataset specific to this research field. It contains 47 tuples (process description, BPMN process model diagram). The biggest strength of this dataset is the wide domain of process descriptions. But a big limitation is the fact that many of the descriptions were translated from other languages to English using tools that are now known as reliable and often underwent slight rewording by the authors, impacting the quality of the descriptions. Furthermore, the model diagrams collected were not in BPMN format; thus, the authors manually remodeled introducing some biases.

Qian et al. (2020) presented a dataset with a limited domain since the entirety of instances, more than 200, refer to recipes. All processes were manually annotated in a three-step process:

1. Identifying each sentence as an action or a statement
2. Assignment of semantics block as begins or ends, and relations as successive, optional, or concurrent to sentences
3. Assignment of the semantic role of words

In conclusion, this dataset does not directly target any process model extraction from a text task. Resulting in being a useful tool for preliminary testing, but not appropriate for experimental evaluation.

Ackermann et al. (2021) propose the *VBP* dataset, a collection of 203 labeled sentences composed of a total of more than 3500 tokens, which often leads to a noisy setting. Each sentence was annotated following different modalities achieved from the Qian et al. (2020) approach, both of which lack details, making this dataset not suitable as a gold standard.

A good comparison of multiple available datasets has been proposed with the *PAGED dataset* (*Du et al., 2024*), which tried to offer a new dataset combining the strengths of the previously developed datasets. A comprehensive overview of the characteristics of each dataset is provided in Table 2. The very high sample size of this dataset made it unsuitable for our analysis, which required direct interaction with the analysis of each process description. However, this dataset, to our knowledge, currently provides the best available gold-standard dataset to evaluate information extraction techniques from NLP.

Dataset	Samples Num	Sequential Actions	Non-sequential Actions			Constraints		Publicly Available
			Exclusive Gateway	Inclusive Gateway	Parallel Gateway	Data Constraint	Action Constraint	
Friedrich et al. (2011)	47	✓	✓	✗	✓	✓	✗	✓
Epure et al. (2015)	34	✓	✓	✗	✓	✗	✗	✗
Ferreira et al. (2017)	56	✓	✓	✗	✓	✗	✗	✗
Mendling et al. (2019)	103	✓	✗	✗	✗	✗	✗	✓
Quishpi et al. (2020)	121	✗	✗	✗	✗	✗	✗	Partial
Qian et al. (2020)	360	✗	✗	✗	✗	✗	✗	✓
Ackermann et al. (2021)	358	✗	✗	✗	✗	✗	✗	✓
López et al. (2021)	37	✓	✗	✗	✗	✗	✗	Partial
Bellan et al. (2023)	45	✓	✓	✗	✓	✗	✗	✓
Liang et al. (2023)	200	✓	✗	✗	✗	✗	✗	✓
Ren et al. (2023)	283	✓	✗	✗	✗	✗	✗	✓
ours	3,394	✓	✓	✓	✓	✓	✓	✓

Table 2: Comparisons between various existing datasets

3 - Extracting Business Processes with LLMs

Many researchers in the field of NLP have been focusing on the extraction of entities and relationships from text, but the extraction of business processes from unstructured data, like textual descriptions, has not received the same level of focus.

On the other hand, Patrizio Bellan has focused his PhD studies on the creation of a database that could fill this void by bridging data-driven information extraction techniques from NLP with the model-driven formalization methods used in Business Process Management (*Process Extraction from Text: a Reference Corpus, a Benchmark, and a Generative AI Approach*, 2024).

Differently from the analysis executed in this project, which focuses on the extraction of declarative constraints, Bellan focused its LLM analysis on the extraction of relevant components of process descriptions, such as activities, authors, and gateways, to create an annotated database for further studies.

This goal has been reached with the creation of the Process Extraction from Text (PET) Dataset, which provides a set of activity labels that exist in the process with a short description that can be used to test various LLMs.

In the next sections, I provided a brief overview of the dataset structure and the prompt used to test the Llama-4-Maverick model on the PET Dataset's analysis.

3.1 - The PET Dataset

The Process Event Template (PET) Dataset represents the first annotated collection of business process descriptions (Bellan 2021) along with comprehensive information about each model. It is used to evaluate LLMs on their ability to perform process discovery from natural language.

It includes a collection of detailed textual descriptions of various process models together with essential elements such as an identifier for activities, actors, activity data, gateways, and their relations. Its developer, Bellan P., made the dataset easily accessible and downloadable at huggingface.co/datasets/patriziobellan/PET.

The dataset contains 45 instances of process description, each of which is represented using 6 values:

- *document name*: a string identifying the document ID;
- *tokens*: a list of strings collecting all the words of the sentence(s) describing the process;
- *token-IDs*: a list of integers representing each word as an integer token;
- *ner-tags*: named entity recognition, a list of strings representing the annotation for each word (see below for more details);
- *sentence-IDs*: an array of integers, starting from 0, representing the number of the sentence within the description, each one separated by a period;
- *relations*: multiple arrays to represent the semantic relations between words, specifying the source and target tokens involved in each one.

Regarding the ner-tags, have multiple options; each token can belong either one of the available chunks:

- | | |
|--|---|
| <ul style="list-style-type: none">• Actor• Activity• Activity data• Further Specification | <ul style="list-style-type: none">• XOR Gateway• Condition Specification• AND Gateway |
|--|---|

Each of the possible identifiers is characterized by additional information, whether the token is at the beginning *B* or inside *I* of the chunk. If the token does not belong to any of these categories, it is identified as *O*.

This dataset was developed to address the lack of a reference dataset essential for comparing data-driven information extraction solutions.

3.2 - Experimental Methodology

To test the abilities of the LLMs regarding the extraction of business processes from unstructured data, we carried out an analysis using the process descriptions of the PET Dataset, testing the ability to extract declarative constraints from this unstructured data.

To analyze the process descriptions, we executed three different approaches, which differ in the volume of information provided in the prompts. A further analysis of the utilized prompts and the information provided is presented in the next section, *3.4 Prompt Design and Description*.

When developing the analysis interface, we evaluated both the use of a local model and the deployment of an **API key**. This last one resulted in better performance in terms of precision and quickness of reply. Furthermore, the usage of the API did not require the download of heavy models, and an easier procedure when switching the model to deploy.

After choosing the use of the API, we compared multiple options, but we finally decided to rely on **Groq** to implement the chat with the LLM. Groq is an online platform that provides a library to support the connection, via API key, to multiple LLMs. This provider was chosen over others based on multiple factors, first of all, its wide range of available models. As a matter of fact, Groq Cloud offers connectivity for Python to over 15 different models and versions:

- qwen-qwq-32b
- deepseek-r1-distill-llama-70b
- gemma2-9b-it
- compound-beta
- compound-beta-mini
- llama-3.1-8b-instant
- llama-3.3-70b-versatile
- llama-guard-3-8b
- llama3-70b-8192
- llama3-8b-8192
- meta-llama/llama-4-maverick-17b-128e-instruct
- meta-llama/llama-4-scout-17b-16e-instruct
- meta-llama/llama-guard-4-12b
- meta-llama/llama-prompt-guard-2-22m
- meta-llama/llama-prompt-guard-2-86m
- mistral-saba-24b

Furthermore, a great characteristic of Groq is a higher token-per-request limit compared to other free services. Lastly, Groq was chosen for its gratuity, which, because of the volume of data analyzed in this project, resulted in it being the best price-quality ratio option. Regarding the **model** from the beginning, we were prone to choosing a Llama model because it represented the optimal model for domain-specific applications due to its strong performance with open access to the model, enabling fine-tuning, transparency, and the possibility of local deployment.

In the first phases of the development of this project, multiple Llama models were tested and compared: Llama-3.1, Llama-4-maverick, and Llama-4-scout. We noticed a clear difference between the results of the 3.1 and the 4 model. The last one offered better interaction, more accurate results, higher precision, memory from previous messages, and an overall better experience. When choosing between the 4-maverick and the 4-scout model, the performances were essentially comparable, so the choice was mainly based on the tokens-per-minute limit. Indeed, there was a ratio of 1 : 5 between the maverick and the scout model. In conclusion, the choice ended up on the 4-scout model.

For each process description, we run an interactive chat developed with the methodology just provided to analyze the unstructured data and **extract the declarative constraints**. In the chat, the LLM was provided with a general overview of the goal and the constraint templates. The LLM received instructions to interact with the user if it was considered necessary to gather further information regarding the process or if any steps were unclear. Each description was analyzed independently and separately from the others to prevent the formation of bias from the precedent processes.

3.3 - Prompt Design and Description

To conduct the analysis of the process descriptions just described, we implemented three distinct approaches, each differing based on the volume of information regarding the constraints' rules provided. In the different approaches, the new information and rules were concatenated with those coming from the previous approach.

In the design process of these prompts, we utilized techniques drawing on both the techniques of *generated knowledge prompting* and the *chain-of-thought* approach, previously introduced in section 2.3.2 - *Prompting Techniques*. The prompts invite the LLM to extract the relevant information regarding activities, authors, and gateways from the textual description of the process and later use this information to tackle the extraction of declarative constraints outlining intermediate reasoning steps necessary for arriving at a solution.

Basic Templates: The first and simplest approach provided the LLM a simple prompt explaining the aim of the chat and the specification of the constraints' rules. For each of the constraint templates, a brief explanation of the rule was provided, together with a couple of examples of situations where the constraint was valid and some other pairs of examples where the constraint was violated.

The templates used in this initial approach were derived from the published article "*Bridging Domain Knowledge and Process Discovery Using Large Language Models*" (Ali Norouzifar et al. 2024). The article provided a list of eight main constraint templates and some general instructions regarding the discovery phase of process mining.

The following is a quote from the initial prompt presenting the aim of the LLM and the example of a declarative constraint template.

Process discovery is a type of process mining aiming to represent the process model explaining how the process is executed. Our focus is on the discovery of an imperative process model. Recently, we developed a framework that allows for some declarative constraints as input to discover better imperative models. Declarative models are close to natural language; however, for a domain expert, they might not be familiar.

...

Consider only the following declarative constraint definitions where a, b, c, and d are activities and template example-template(x,y) specifies a template named "example-template" and x and y can be one of the activities from the set of all activities. Do not reason based on the template names and only use explanations to understand the meaning of specific templates:

1. *at-most(A): A occurs at most once.*

a. *Some examples satisfying this constraint:*

i. *The process starts with activity B for setup, followed by A for approval, then C for data collection, and ends with D for implementation. Optional execution of B can occur again before C.*

ii. *The workflow begins with C for initial data collection, followed by D for setup, then A for mandatory approval, and concludes with B for detailed planning.*

iii. *The process initiates with D for setup, followed by B for planning, C for data analysis, and finally A for essential approval. Optional execution of C may occur again after B.*

b. *Some examples violating this constraint:*

i. *The process starts with A for initial approval, followed by B for planning, then loops back to A for re-approval, and ends with C for data analysis and D for implementation.*

ii. *Begin with C for data collection, followed by A for approval, then B for planning, another A for re-approval, and ends with D for execution.*

iii. *The workflow involves D for setup, followed by parallel execution of A for initial approval and B for planning, then A second occurrence of A for final approval, and C for final data analysis.*

Additional Templates: In the second approach, the goal was to expand the pool of Declare templates, so we referenced the constraints defined in the “*Outcome-Oriented Prescriptive Process Monitoring Based on Temporal Logic Patterns*” (Donadello et al, 2023) to provide seventeen additional constraint templates. Similarly, as in the previous prompt, each new template is provided with a brief explanation of the rules and examples that respect and violate the rules. An overview of the templates is provided in Table 3.

Family: Template	LTL _f Semantics	Description
\mathcal{E} : EXISTENCE (n, A)	$F(A \wedge X\text{existence}(n-1, A))$	A has to occur at least n times.
\mathcal{E} : ABSENCE (n + 1, A)	$\neg\text{existence}(n+1, A)$	A has to occur at most n times.
\mathcal{E} : EXACTLY (n, A)	$\text{existence}(n, A) \wedge \text{absence}(n+1, A)$	A has to occur exactly n times.
\mathcal{E} : INIT (A)	A	Each case has to start with A.
\mathcal{C} : CHOICE (A, B)	$FA \vee FB$	A or B have to occur at least once.
\mathcal{C} : EXCLUSIVE CHOICE (A, B)	$(FA \wedge \neg FB) \vee (\neg FA \wedge FB)$	A or B have to occur at least once but not both.
\mathcal{PR} : RESPONDED EXISTENCE (A, B)	$FA \rightarrow FB$	If A occurs, B must occur as well
\mathcal{PR} : RESPONSE (A, B)	$G(A \rightarrow FB)$	If A occurs, B must eventually follow.
\mathcal{PR} : ALTERNATE RESPONSE (A, B)	$G(A \rightarrow X(\neg AUB))$	If A occurs, B must eventually follow without any other A in between.
\mathcal{PR} : CHAIN RESPONSE (A, B)	$G(A \rightarrow XB)$	If A occurs, B must occur next.
\mathcal{PR} : PRECEDENCE (A, B)	$(\neg BUA) \vee G(\neg B)$	B can occur only if A has occurred before.
\mathcal{PR} : ALTERNATE PRECEDENCE (A, B)	$(\neg BUA) \wedge G(B \rightarrow X((\neg BUA) \vee G(\neg B)))$	B can occur only if A has occurred before, without any other B in between.
\mathcal{PR} : CHAIN PRECEDENCE (A, B)	$G(X(B) \rightarrow A)$	B can occur only immediately after A.
\mathcal{NR} : NOT RESPONDED EXISTENCE (A, B)	$FA \rightarrow \neg FB$	If A occurs, B cannot occur.
\mathcal{NR} : NOT RESPONSE (A, B)	$G(A \rightarrow \neg(FB))$	If A occurs, B cannot eventually follow.
\mathcal{NR} : NOT PRECEDENCE (A, B)	$G(FB \rightarrow \neg A)$	A cannot occur before B.
\mathcal{NR} : NOT CHAIN RESPONSE (A, B)	$G(A \rightarrow X(\neg B))$	If A occurs B cannot occur next.
\mathcal{NR} : NOT CHAIN PRECEDENCE (A, B)	$G(XB \rightarrow \neg A)$	A cannot occur immediately before B.

Table 3: LTL_f semantics and textual description of the declare templates

In this prompt, the modalities to present each template is analogous to the first prompt.

4. *not-response(A,B): If A happens, then B must not happen afterward. B is not allowed after A.*

a. *Some examples satisfying this constraint:*

- i. *If an invoice is canceled (A), no payment (B) should happen after.*
- ii. *The technician closes the ticket (A) and archives it without any further action.*
- iii. *After closing the ticket (A), the system logs the case as resolved with no reopening (B).*
- iv. *A ticket is closed (A) and followed by a feedback request, with no reopen (B) in the workflow.*
- v. *The sequence B □ A □ C is allowed*

b. *Some examples violating this constraint:*

- i. *The ticket is closed (A) and then later reopened (B) due to a missed issue.*
 - ii. *Closing (A) is followed by a user complaint, leading to a reopen (B).*
 - iii. *After a ticket is closed (A), a second-level agent reopens it (B) for escalation.*
-

Hierarchical relations: Finally, the third approach aimed at filtering the found constraints to represent the maximal number of relations with the least number of constraints. In the paper “*Patterns for a Log-Based Strengthening of Declarative Compliance Models*” (Schunselaar, Maggi, and Sidorova, 2012), multiple dependencies between constraints are defined, which are represented in Figure 4. In our prompt, only those relevant to the previously defined constraints were provided to the LLM. That is, those regarding the following constraints:

- Chain-Succession
- Chain-Response
- Alternate-Response
- Response
- Responded-Existence
- Chain-Precedence
- Alternate-Precedence
- Precedence
- Alternate-Succession
- Succession
- Co-Existence
- Init
- Exclusive-Choice
- Choice
- Not-Exclusive-Choice
- Not-Succession
- Not-Chain-Succession

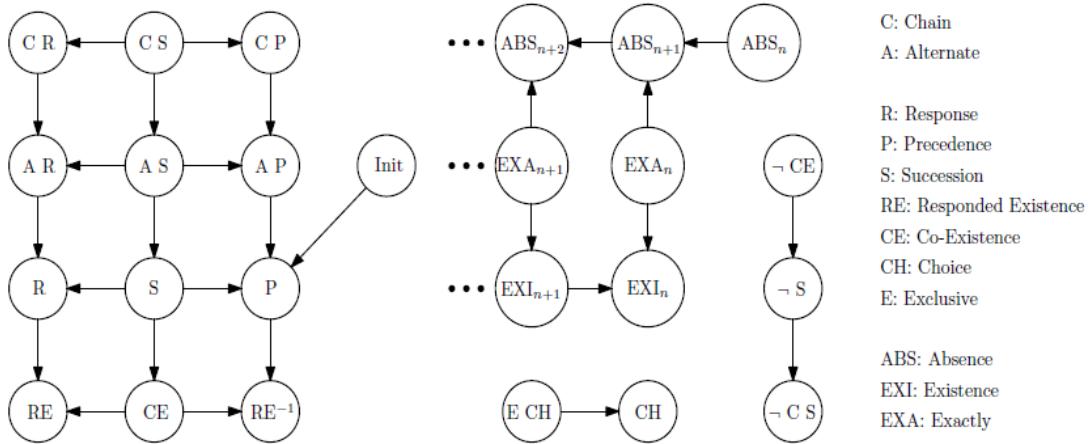


Figure 4: The hierarchy of the Declare constraints

The LLM is prompted to represent only the constraints found on the higher level of the hierarchical ladder.

These constraints are related together through hierarchical relationships. If a constraint implies one or more other constraints, only the strongest (highest level) should be present in the final constraint list

Furthermore, a general prompt regarding the formatting specifications of the results has been provided for each approach. The LLM received some instructions on how to set up the response in order to make it easier to extract the final declarative constraint and the activities identified in the process. The following is a citation from this last prompt.

... In our final interaction the constraint should be defined as "Final Formal Declarative Constraints: " (this line MUST appear only once in your reply, if you have any intermediate interpretations you can use "Temporary Declarative Constraints" or similar phrases) followed by one constraint per line defined using the template provided in the previous instruction.

Furthermore, in a line, you have to summarize all the activities present in the constraints, writing it as "Activities: " and on the same line, all the activities divided by commas, you cannot put a dot at the end of the line

Be succinct and do not hallucinate.

All prompts utilized across the various approaches are detailed in the appendix.

3.4 - Integration in Declare4Py

After the positive performance evaluation of the usage of the LLM in the constraints' extraction in the PET Dataset, we implemented this concept in the Declare4Py library, previously presented, to further expand its discovery task capabilities. The library is freely accessible via <https://github.com/ivanDonadello/Declare4Py>.

To implement the possibilities to extract the constraints from a textual description, we created a new class called `TextualModel`, a child of the `ProcessModel` abstract class.

Similarly, as in the analysis of the PET dataset methodology, we evaluated the usage of a local model or an API key. Our decision to deploy an AIP key was driven by the following aspects. Firstly, the usage of an API key allows the user to work with online models without requesting to download heavy models directly on the user's machine. Furthermore, this allows for the execution of the analysis with the latest version of the model up to the state of the art. In addition, the deployment of an API key accelerates the execution because the evaluation is not performed on the user's machine, drastically reducing the latency time. However, this presents one limitation: the necessity of being online which increases the risk of exposure at cyber-attacks.

The `Declare4Py.ProcessModels`.`The TextualModel` class is responsible for creating a model whose declarative constraints and activities are extracted from a textual description of the process via interaction with the LLM.

These objects are initialized by importing a textual description of the model as a string or a .txt file, and later the constraints and actions are extracted from the description using the `to_decl` method.

We **initialize** a `TextualModel` object the textual description of the process is required. The textual description can be directly passed as a parameter in the `constructor` or later uploaded from a textual file using the `parse_form_file method`.

The main method of the `TextualModel` class is the **`to_decl` method**, which handles the whole discovery process.

The method's signature identifies three *parameters*, two of which are optional:

- *API key*: is a string variable that is used to set up the environment variable for Groq connectivity
- *interactive*: is a Boolean value which identifies whether or not the chat will be interactive with the user, or not; by default, it is set as `False`
- *llm_model*: is a string variable that identifies the exact model of the LLM; by default, it uses the `llama-4-scout`.

For all parameters, if the passed value provided by the user is incompatible with the possible values, the situation is handled by setting the default value or throwing an error.

In the planning phase of the analysis, we wanted to ensure the user could both interact with the LLM if any clarification regarding the process description was needed, but also not interact whenever time was not available or whenever analyzing multiple processes. For this reason, the *Interactive parameter* allows the user to choose whether she/he would like to interact with the chat. In the case where the interaction is turned off, by setting the parameter as `False`, the LLM will be prompted to directly return its response, extracting the constraints and the activities without any user intervention. On the other hand, when turned on, the interaction will be active, and when necessary, the LLM will open a chat with the user posing requests and specifications regarding the process description.

The available *model names* can be found both directly on Groq Cloud and within the code.

The `to_decl` method starts its execution by verifying all parameters provided by the user. Next, based on the value of the *interactive* parameter, the LLM is prompted slightly differently.

If the *interaction is on*, the LLM receives the prompts and a chat with the user is started; if the LLM considers it necessary to request additional information, it will do so and the user will have the chance to interact with the model as long as she/he require. Once the user is satisfied with the LLM's response, it can "confirm" the response and proceed to the next phase.

On the other hand, if the *interaction is turned off*, the LLM is prompted to directly define the constraints within its first response, and the process then goes directly to the next step.

All prompts provided to the LLM can be found in the Declare4Py repository in the following folder: <https://github.com/ivanDonadello/Declare4Py/blob/main/Declare4Py/Utils/DeclarePrompts.py>.

Once the final LLM response has been retrieved, multiple support methods are used to extract the constraints and activities of the process. This phase is facilitated by instructing the LLM to use a specific predefined structure to present the final version of the extracted declarative constraints.

Next, the extracted lists of constraints and activities are formatted as a string following the syntax of the .decl files. This string is used to convert into a DeclareModel object using the parse_from_string method of this last class. Finally, the model is **returned** and can be further analyzed or simply saved to a file.

To utilize the TextualModel class and its analysis method, it is necessary to set up the **Groq API key**. A clear and brief step-by-step explanation is present in the Tutorial of the TextualModel class, together with a more in-depth presentation of the class and sample code, at the following link:

<https://github.com/ivanDonadello/Declare4Py/blob/main/docs/source/tutorials/10.TextualModel.ipynb>

More information regarding the role of each method and variable can be found directly within the code accessible at the following link:

<https://github.com/ivanDonadello/Declare4Py/blob/main/Declare4Py/ProcessModels/TextualModel.py>.

3.5 - Experimental Results

Unfortunately, the absence of a ground truth on declares constraints or activity names for PET sentences makes a complete **performance evaluation** using precision and recall metrics impossible; therefore, we limit ourselves to a quantitative evaluation of three metrics:

1. *The number of user interactions*
2. *The volume of extracted declarative constraints*
3. *The volume of identified activity names*

What we observed has been summarized in Table 4, providing the maximal and average values for these three parameters.

Approach	Max Interactions	Avg. Interactions	Max Constraints	Avg. Constraints	Max Activity	Avg. Activity
<i>Basic templates</i>	1	0,11	30	10,67	23	8,16
<i>Additional templates</i>	0	0	34	11,36	30	9,04
<i>Hierarchical Rules</i>	0	0	19	8,76	15	8,11

Table 4: Overview of analysis results

The first observation that we can derive is regarding the *Interaction count*. As a matter of fact, only in the first approach, where the constraints templates were minimal, did the chat interact with the user, but only in a very small number of instances. Only circa 11% of the processed required further information before being able to provide a final version of the declarative constraints. In the second and third approaches, where additional templates and information regarding hierarchical rules among the constraints were respectively provided, the interaction was absent in 100% of the processes. This analysis leads us to the conclusion that the more information and templates are provided, the better the LLM is capable of interpreting the process descriptions by categorizing each sentence and relationship in the form of a constraint.

A second reflection can be made on the *extracted declarative constraints*. As expected, the volume of extracted constraints increases from the first to the second approach and decreases in the third and last approach. This was expected because, from the first to the second approach, the volume of templates provided in the LLM instructions increased from 8 to 25 constraints, allowing the model to better identify each relation. In the last approach, the LLM is prompted to report only the essential constraints, utilizing the hierarchical rules representing only the highest constraints in hierarchical rank.

Finally, the maximal *activity count* slightly increases from the first to the second approach and drastically decreases from the second to the third approach. This is most likely caused by the wrong interpretation of the prompt by the LLM, which probably took the addition of new constraints, in the second approach, as a general instruction to increase the details of the activities. And in the third approach, the instructions to reduce ambiguous and repetitive constraints were presumably taken as a general instruction to reduce also the repetitive activities. However, the average volume is maintained pretty much constant throughout the three approaches, leading us to the conclusion that the overall result is coherent across all the approaches.

Another obstacle to the analysis of results is the incompatibility of activity names that are represented differently in each approach. In fact, each analysis abbreviates activities differently by using performers and actions in some cases and including the recipient in others. In addition, actions are often declined differently, in the present tense, past tense, present continuous, or even in the gerund.

The complete dataset, the prompt used in the analysis, the interactions, and the parsed results are available at the following repository: https://github.com/IreneAvezzu/PET_Dataset_Analysis.

Aiming for a clearer and more precise view of the performances of the different approaches and the LLMs' usage, we performed a qualitative analysis on three processes described in the PET Dataset. The results of the case study carried out on this smaller sample are widely illustrated and explained in Chapter 4.

4 - Case Studies

As previously discussed in section 3.3 *Experimental Methodology*, due to the absence of ground truth, we were unable to provide a quantitative analysis of the results and the LLM extraction capabilities on the overall dataset. For this reason, we decided to perform a more comprehensive analysis on a smaller sample of the process descriptions provided by the PET Dataset.

Prior to a comprehensive evaluation of the results and performance of the LLM in extracting declarative constraints from a textual description, we would like to highlight an observation concerning the volume of information extracted.

Examining the volume of the results of the extraction process of all process descriptions, it is clear that the quantity of the extracted information aligns with the expectations outlined in section 3.3 *Experimental Methodology*. Specifically, the number of extracted constraints increases from the first to the second approach, then remains steady or decreases from the second to the third approach. This pattern is consistent with our earlier predictions; indeed, the second approach introduces 17 new templates, while the third approach enforces the representation of only those constraints that are higher in the hierarchy, excluding the implied rules from the final output.

Regarding activities, we predicted that the quantity would either increase or stay consistent as directions were incorporated into the model. The resulting data shows that the volume of extracted activities aligns with these expectations.

In the following sections are provided firstly a general overview of the set-up of the case studies and the complete analysis of each process is provided.

4.1 - Experiments Set Up

In this section are presented the results of the analysis of the extraction process of the LLM on three selected process descriptions provided in the PET Dataset are presented. The processes have been chosen to present a general overview of the characteristics of the process descriptions and the possible results of the interaction.

Each process description has been analyzed using an *interactive approach*, offering the LLM the possibility to interact with the user, requesting additional information. However, this rarely happened.

The chosen descriptions are those corresponding to the processes' numbers 15, 35, and 43 identified by the document names: doc-9.4, doc-9.5, doc-10.11.

These textual descriptions were picked on the grounds of their length, indeed the descriptions' length range spans between 29 and 697 words, with an average of circa 150 words. These process descriptions, whose lengths are 98, 117, and 72, respectively, represent an adaptive intermediary value.

Furthermore, doc-9.5 has been chosen because it represents one of the few processes that, during the first analysis approach, the LLM requested additional information regarding the process.

In the following sections, divided by process, are presented the textual descriptions on which the constraint extraction was performed, directly obtained from the PET dataset, a flow chart of the process that I personally developed using BPMN rules to visualize the possible path of execution of the process, two tables representing a summary of the volume of the results of the extraction processes, and the complete list of extracted constraints and activities, and lastly a qualitative evaluation of the results.

Additionally, thanks to the visual representation of the model, we were able to define a ground truth to compare the results extracted by the LLM, highlighting the constraints that should have been defined by every approach based on the instructions provided in the prompt; we developed it keeping in consideration the activities identified in each approach. These expected results are not singularly analyzed within each process' section but as a whole in the final section 4.5 - *Evaluation*.

The complete LLM interaction for every approach can be found in the appendix.

4.2 - Case Study with Process 15

The following is the process description of the document doc-9.4 from the PET dataset, and in Figure 5, the process is visually represented in its entirety.

Once the dates are finalized (by the Coordination Unit), the Support Officer updates all group calendars and creates meeting folders for each meeting and ensures all appropriate documents are uploaded to system. Committee Members are advised a week before each meeting to read all related documents. The Committee Members hold their meeting, and the Support Office then produces minutes including any Action Points for each Committee Member. Within 5 working days, the Coordination Unit must conduct a QA check on the minutes, which are then sent to all Committee Members. The Support Officer then updates all departmental records.

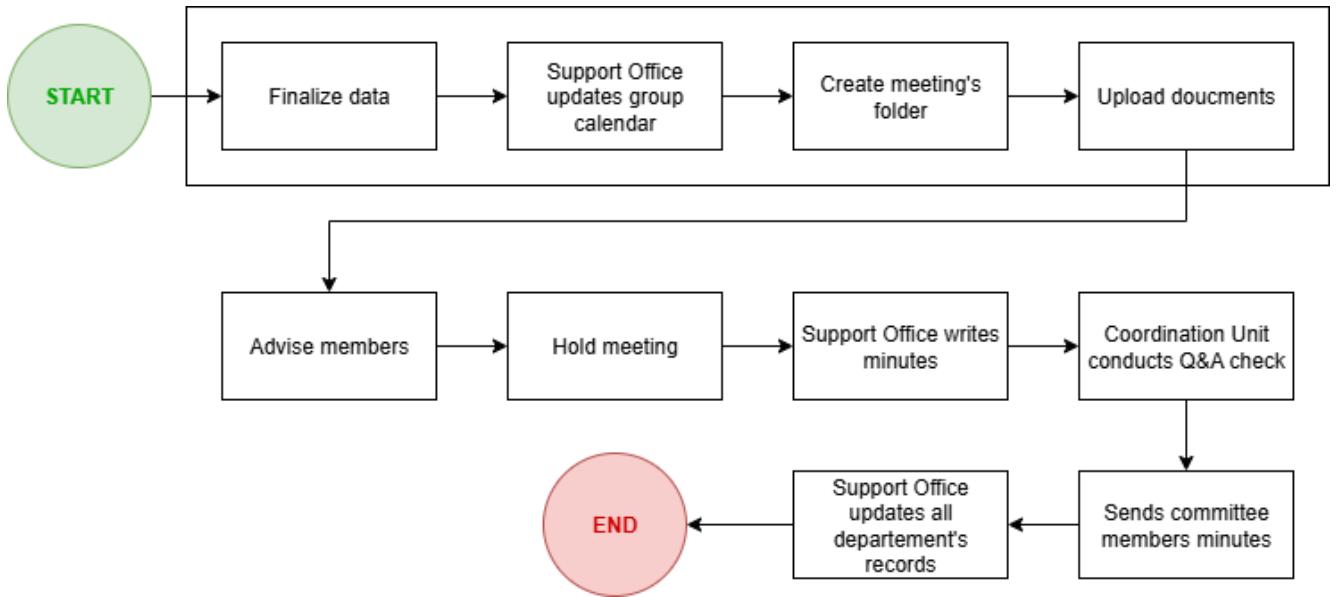


Figure 5: Visual representation of doc-9.4

Process number 15 is certainly, among the three processes analyzed, the one characterized by the simplest and most direct flow. Indeed, there are neither loops nor XOR gates within the model.

Let us start by examining the **activities**, as already anticipated, the volume is slightly lower in the first approach (6 activities detected), and then increases in the second approach. In the third approach, the identified activities are the same as in the second phase, and identify all the activities presented in the textual description. In the first approach, the activities related to the first and last steps of the process are not detected; this could have been detected using the **existence(A)** constraint defined as follows in the prompt:

existence(A): A occurs at least once, which means the existence of activity A is mandatory.

We now proceed by presenting the observations derived from observing the declarative constraints extracted in the three consecutive approaches.

Thanks to the visual representation, Table 5 highlights the constraints that should have been defined by every approach based on the instructions provided in the prompt; the identified activities were kept in consideration in the development of the constraints.

Basic templates	Additional templates	Hierarchical Rules
<p>1. at-most(...) for each of the 6 identified activities</p> <p>2. existence(...) for each of the 6 identified activities</p> <p>3. response(update_calendar, create_folder)</p> <p>4. response(create_folder, hold_meeting)</p> <p>5. response(hold_meeting, write_minutes)</p> <p>6. response(write_minutes, QA_check)</p> <p>7. response(QA_check, send_minutes)</p> <p>8. precedence(update_calendar, create_folder)</p> <p>9. precedence(create_folder, hold_meeting)</p> <p>10. precedence(hold_meeting, write_minutes)</p> <p>11. precedence(write_minutes, QA_check)</p> <p>12. precedence(QA_check, send_minutes)</p> <p>13. responded-existence(update_calendar, create_folder)</p> <p>14. responded-existence(create_folder, hold_meeting)</p> <p>15. responded-existence(hold_meeting, write_minutes)</p> <p>16. responded-existence(write_minutes, QA_check)</p> <p>17. responded-existence(QA_check, send_minutes)</p>	<p>1. at-most(...) for each of the 10 identified activities</p> <p>2. existence(...) for each of the 10 identified activities</p> <p>3. response(...) from basic templates approach</p> <p>4. response(finalize_data, update_calendar)</p> <p>5. response(create_folder, update_documents)</p> <p>6. response(update_documents, advise_members)</p> <p>7. response(send_minutes, update_records)</p> <p>8. precedence(...) from basic templates approach</p> <p>9. precedence(finalize_data, update_calendar)</p> <p>10. precedence(create_folder, update_documents)</p> <p>11. precedence(update_documents, advise_members)</p> <p>12. precedence(send_minutes, update_records)</p> <p>13. responded-existence(...) for each pair of consecutive actions</p> <p>14. Init(finalize_data)</p> <p>15. succession(...) for each pair of consecutive actions</p> <p>16. non-exclusive-choice(...) for each pair of consecutive actions</p> <p>17. chain-succession(...) for each pair of consecutive actions</p> <p>18. chain-response(...) for each pair of consecutive actions</p> <p>19. chain-precedence(...) for each pair of consecutive actions</p>	<p>1. at-most(...) for each of the 10 identified activities</p> <p>2. existence(...) for each of the 10 identified activities</p> <p>3. Init(finalize_data)</p> <p>4. chain-succession(...) for each pair of consecutive actions</p>

Table 5: Expected declarative constraints for each approach analysis

The observation of the constraints extracted from the **basic approach** leads us to recognize how it manages to describe the succession of all activities, even though the initial and concluding activities of the process are not recognized. However, comparing the extracted to the expected constraints, we observe how the majority of at-most, existence, precedence, and response constraints are not identified, and the pair of consecutive activities is described by either a precedence or a response constraint. Furthermore, none of the responded-existence constraints have been identified.

Referring to the constraints extracted from the **approach with additional templates**, we notice how one of the problems of the previous step is solved. Indeed, here the extracted constraint list contains *Init(finalize_dates)*, which correctly recognizes the activity that starts the process. Similarly to before, in this second approach, none of the constraints of type at-most, existence, and responded-existence have been identified. The same issue appears related to the constraints of type non-exclusive-choice, chain-succession, chain-response, and chain-precedence. Furthermore, most pairs of consecutive activities are described by either a precedence, response, or succession constraint.

Finally, inspecting the constraints extracted from the **approach with hierarchical rules**, we observe that, similar to the previous approaches, the process flow from start to finish is recognized, but in this case, the process is conceptually represented in two separate blocks; this is also represented in the process model. Unlike the other approaches, in this case, several declarative constraints of the existence type are implemented, already defined since the first approach. In this approach, the activities of *finalize_dates*, *hold_meeting*, *produce_minutes*, and *conduct_qa_check* are identified as mandatory. Specifically, the *finalize_dates* activity is redundantly represented by both the existence-type constraint and the Init constraint; this is an unnecessary repetition. However, the LLM correctly behaved based on the provided instruction since the exists-type constraints do not appear nor have a role in the hierarchical ladder.

As in the previous approach, comparing the extracted to the expected constraints, we observe the absence of most existence constraints and the entirety of at-most constraints. Furthermore, we expected each pair of consecutive activities to be represented in a chain-succession type constraints, since this would be the highest type of constraints with respect to what was defined by the hierarchical rules; however, no constraints of this type were identified.

Tables 6 and 7 that follow this paragraph are the representation of the results of the extraction process from the textual description.

Approach	Extracted Constraints	Extracted Activities	Interaction
<i>Basic templates</i>	7	6	No
<i>Additional templates</i>	13	10	No
<i>Hierarchical Rules</i>	13	10	No

Table 6: Volume of results of the extraction process of doc-9.4

Approach	Extracted Constraints	Extracted Activities
<i>Basic templates</i>	<ol style="list-style-type: none"> 1. at-most(conduct_qa_check) 2. existence(hold_meeting) 3. response(update_group_calendars, create_meeting_folders) 4. response(hold_meeting, produce_minutes) 5. response(produce_minutes, conduct_qa_check) 6. response(conduct_qa_check, send_minutes) 7. precedence(conduct_qa_check, send_minutes) 	<ol style="list-style-type: none"> 1. hold_meeting 2. create_meeting_folders 3. update_group_calendars 4. send_minutes 5. produce_minutes 6. conduct_qa_check
<i>Additional templates</i>	<ol style="list-style-type: none"> 1. response(finalize_dates, upload_documents) 2. response(upload_documents, advise_committee_members) 3. response(advice_committee_members, hold_meeting) 4. response(hold_meeting, produce_minutes) 5. response(produce_minutes, conduct_qa_check) 6. response(conduct_qa_check, send_minutes) 7. response(send_minutes, update_departmental_records) 8. precedence(finalize_dates, update_group_calendars) 9. precedence(finalize_dates, create_meeting_folders) 10. Init(finalize_dates) 11. succession(produce_minutes, conduct_qa_check) 12. succession(conduct_qa_check, send_minutes) 13. succession(send_minutes, update_departmental_records) 	<ol style="list-style-type: none"> 1. hold_meeting 2. create_meeting_folders 3. update_departmental_records 4. update_group_calendars 5. send_minutes 6. produce_minutes 7. upload_documents 8. conduct_qa_check 9. advise_committee_members 10. finalize_dates
<i>Hierarchical Rules</i>	<ol style="list-style-type: none"> 1. existence(finalize_dates) 2. existence(hold_meeting) 3. existence(produce_minutes) 4. existence(conduct_qa_check) 5. response(advice_committee_members, hold_meeting) 6. precedence(produce_minutes, conduct_qa_check) 7. Init(finalize_dates) 8. succession(finalize_dates, update_group_calendars) 9. succession(update_group_calendars, create_meeting_folders) 10. succession(create_meeting_folders, upload_documents) 11. succession(hold_meeting, produce_minutes) 12. succession(conduct_qa_check, send_minutes) 13. succession(send_minutes, update_departmental_records) 	<ol style="list-style-type: none"> 1. hold_meeting 2. create_meeting_folders 3. update_departmental_record 4. update_group_calendars 5. send_minutes 6. produce_minutes 7. upload_documents 8. conduct_qa_check 9. advise_committee_members 10. finalize_dates

Table 7: Results of the extraction process of doc-9.4

4.3 - Case Study with Process 35

Before presenting the results and the analysis of the process doc-9.5 from the PET dataset, we introduce its process description, and, with Figure 6, the process is visually represented in its entirety.

After the Expense Report is received, a new account must be created if the employee does not already have one. The report is then reviewed for automatic approval. Amounts under \$200 are automatically approved, whereas amounts equal to or over \$200 require approval of the supervisor. In case of rejection, the employee must receive a rejection notice by email. Otherwise, the reimbursement goes to the employees direct deposit bank account. If the request is not completed in 7 days, then the employee must receive an approval in progress email. If the request is not finished within 30 days, then the process is stopped and the employee receives an email cancellation notice and must re-submit the expense report.

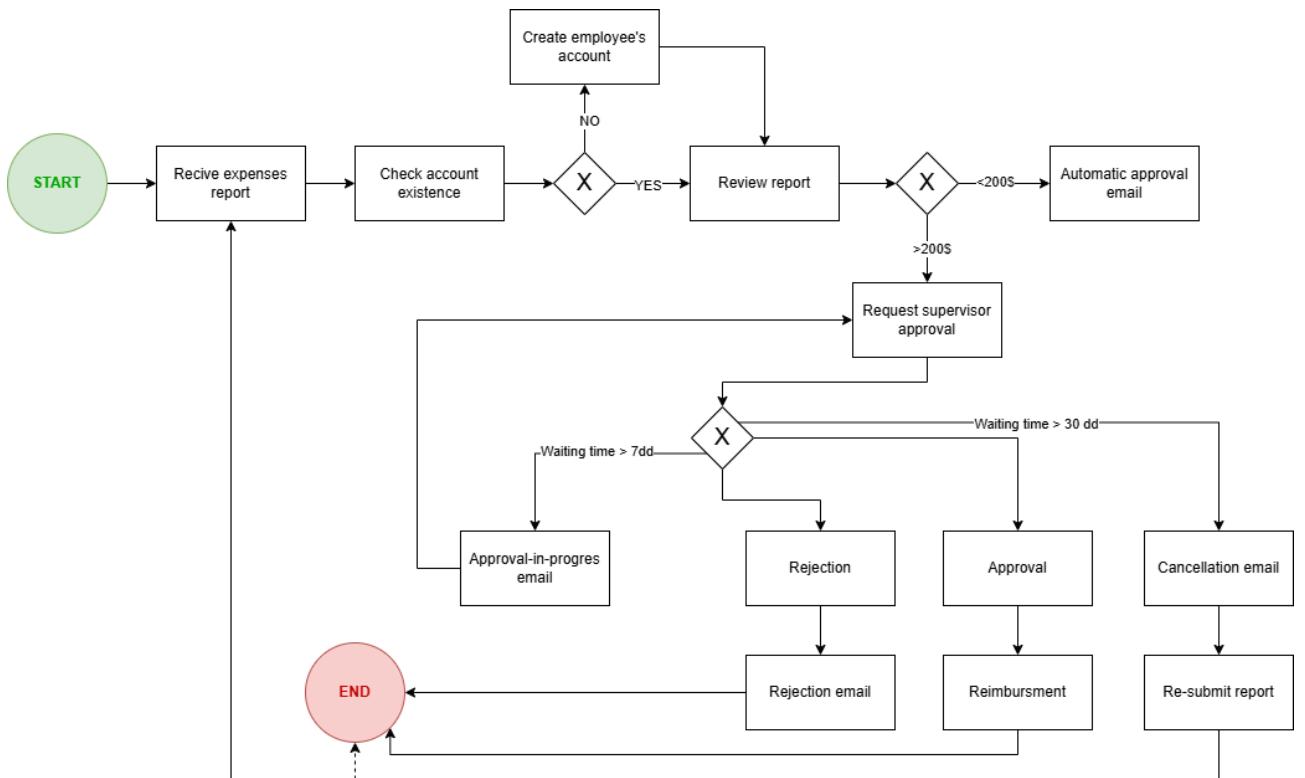


Figure 6: Visual representation of doc-9.5

This procedure, in contrast to the previous one, presents some decision points that require the selection of a specific direction over others. Additionally, it features some loops that complicate the representation of the process through declarative templates.

This process, as introduced before, belongs to the 11% of processes that, in the analysis conducted with the first and most simple approach, required further clarification. In this case, the LLM requested more information regarding two specific activities: create an employer account and re-submit the folder. The requests cover the compulsory nature of the presence of these activities, or instead if they are supplementary actions.

... Also, I have a question: are "create_account" and "re_submit_report" part of the main process flow or are they optional? The text mentions that a new account must be created if the employee does not already have one, but it's not clear if this is a separate process or a step that always happens. Similarly, "re_submit_report" seems to be a loop back to the beginning of the process, but I'd like clarification on how it's triggered...

To respond to these questions, we reviewed the model schema, presented in section 4.1 *Experiment Set Up*. Based on our interpretation, we considered the creation of an employer's account to be an optional step necessary only if the control of the presence of an account returns negative. Similarly, the re-submission of the process is optional, if after a long wait the employer wants to re-present its expenses report, which is most likely going to happen every time since we are analysing the case in which the amount is over 200\$. The completion of these activities brings the process execution back to the starting point.

"Create_account is an optional step, so you can decide if include it or not in the constraints. re_submit_report creates a loop going back to the point where a submit_report action happened, but this action "re_submit_report" can only happen after the request is not finished within 30 days and the client has received email cancellation notification."

Starting from the list of extracted **activities**, it is observed that the first approach identifies eight which constitute the core of the process. However, similar to the analysis conducted on the previous process, the activity that serves as the starting point of the process is also not recognized in this instance.

The second and third approaches enlarge the activities list, bringing it to a total of 8 actions, although they are slightly different.

The second approach distinguishes itself from the successor by defining the activity of *resubmit_expense_report*, which creates a possible loop within the execution of the process. This is the sole case in which this activity is acknowledged; in contrast, the other approaches the LLM defines each possible repetition of the report analysis independently of prior occurrences. Both approaches are logically and syntactically valid.

The third approach incorporates activities such as *notcompletedwithin7days* and *notfinishedwithin30days* that address the time-related controls specified in the textual description of this process.

A noteworthy observation in the analysis of the activities is how the major activities represented in the process flow model are identified by all three approaches. However, there are certain actions, such as *check_employer_account* and *ask_supervisor*, that remain unidentified in any of the approaches.

Continuing with the analysis, we examined the constraints extracted utilizing the three different analytical methods. Similarly to how we proceeded in the analysis in Process 15, we used the visual representation of the process to define the ground truth. The constraints are illustrated in Table 8.

Basic templates	Additional templates	Hierarchical Rules
<ul style="list-style-type: none"> 1. at-most(...) for each of the 8 identified activities 2. existence(...) for each of the 8 identified activities 3. response(approval, reimbursement) 4. response(cancellation_email, re_submission) 5. precedence(...) for each pair of activities related by response constraints 6. co-existence(...) for each pair of activities related by response and/or precedence constraints 	<ul style="list-style-type: none"> 1. at-most(...) for each of the 11 identified activities 2. existence(...) for each of the 11 identified activities 3. response(approval, reimbursement) 4. response(rejection, rejection_email) 5. response(cancellation_email, re_submission) 6. precedence(...) for each pair of activities related by response constraints 7. precedence(recv_report, review_report) 8. precedence(recv_report, supervisor_approval) 	<ul style="list-style-type: none"> 1. at-most(...) for each of the 11 identified activities 2. existence(...) for each of the 11 identified activities 3. chain-succession (approval, reimbursement) 4. chain-succession (rejection, rejection_email) 5. chain-succession (cancellation_email, re_submission) 6. chain-succession (time_over_7dd, approval_in_progress_email) 7. chain-succession (recv_report, review_report)

<p>7. not-co-existence (supervisor_approval, rejection_email)</p> <p>8. not-succession (supervisor_approval, rejection_email)</p> <p>9. responded-existence(...) for each pair of activities related by response constraints</p>	<p>9. precedence(receive_report, rejection)</p> <p>10. precedence(receive_report, approva_in_progress_email)</p> <p>11. precedence(receive_report, cancellation_email)</p> <p>12. co-existence(...) for each pair of activities related by response constraints</p> <p>13. not-co-existence (supervisor_approval, rejection_email)</p> <p>14. not-succession (supervisor_approval, rejection_email)</p> <p>15. responded-existence(...) for each pair of activities related by response constraints</p> <p>16. Init(receive_report)</p> <p>17. not-responded-existence(...) for each pair of activities related by not-co-existence constraints</p> <p>18. not-response (...) for each pair of activities related by not-co-existence constraints</p> <p>19. not-precedence(...) for each pair of activities related by not-co-existence constraints</p> <p>20. succession(...) for each pair of activities related by response constraints</p> <p>21. not-exclusive-choice(...) for each pair of activities related by response constraints</p> <p>22. chain-succession(...) for each pair of activities related by succession constraints</p> <p>23. chain-response(...) for each pair of activities related by response constraints</p> <p>24. chain-precedence(...) for each pair of activities related by precedence constraints</p>	<p>8. chain-succession (receive_report, supervisor_approval)</p> <p>9. chain-succession (receive_report, rejection)</p> <p>10. chain-succession (receive_report, approva_in_progress_email)</p> <p>11. chain-succession (receive_report, cancellation_email)</p> <p>12. chain-succession (receive_report, create_account)</p> <p>13. not-co-existence (supervisor_approval, rejection_email)</p> <p>14. not-succession (supervisor_approval, rejection_email)</p> <p>15. Init(receive_report)</p> <p>16. not-response (...) for each pair of activities related by not-co-existence constraints</p> <p>17. not-precedence(...) for each pair of activities related by not-co- existence constraints</p> <p>18. not-exclusive-choice(...) for each pair of activities related by response constraints</p>
--	---	--

Table 8: Expected declarative constraints for each approach analysis

In the **basic extraction method**, the process flow is represented exclusively from the moment when the report is reviewed, omitting the previous execution block in which the report is received, and verification of the employee's account occurs.

The XOR gateway related to report rejection and approval is accurately represented through a declarative constraint of type *not-co-existence*. The exclusive choice between the reimbursement, performed as a result of approval, and the waiting period exceeding thirty days that leads to the cancellation of the review, is appropriately illustrated by a constraint of type *not-succession*. Unlike the two cases just presented, the point of choice concerning *automatic_approval* versus *supervisor_approval* is not represented in any manner.

Furthermore, similarly to the previously analyzed process, multiple instances are noted where the same pair of activities is defined by both a *precedence* and *response* type constraint, leading to redundancies in the model.

In this approach, two issues can be identified. The first concerns the *existence-type* constraint related to *cancellation_email*, according to the process description, is only necessary if the review is suspended after thirty days from the initial supervisor's request for approval. However, this constraint enforces its necessity in every execution, contradicting the correctly defined XOR gateways.

Additionally, there is a misinterpretation of the process by the LLM, which does not align with the specifics defined in the description. Indeed, in the final step, constraints are introduced that stipulate that resubmission can only be performed following rejection. This is incorrect; it is only permissible following a suspension due to an excessively long waiting period.

In the **methodology with additional templates**, the initial operation of the process is accurately identified. Similar to the first approach, the XOR gateway concerning the rejection and approval of the report is recognized; however, in this case, it is represented with a *non-response* type constraint.

The choice related to *automatic_approval* and *supervisor_approval* is not represented in this extraction, as was the case in the previous approach.

The first issue identified in this process of extracting declarative constraints is the improper and excessive application of *existence-type* constraints. Among the *existence-type* constraints extracted, it is evident that they should have been represented as *at-most-type*, as they represent operations executed across parallel potential execution streams. The syntax indicates that the activities enclosed within the unary constraint of type *existence* must be executed at least once; otherwise, they are mandatory. In contrast to what is described in the textual description, i.e., that each of these actions excludes the others.

The second error identified is the incorrect definition of the *not-succession(review_report, reimbursement)* constraint, which, although it is defined correctly according to the template's syntax, does not comply with what the process description describes, which states that reimbursement is one of the potential actions that may conclude the process after the report review.

The **hierarchical approach** effectively addresses the issues that appeared in the second methodology.

Here, the XOR gateways associated with the method of approval (automatic vs. supervisor) and the possible outcomes of the process (rejection-reimbursement and cancellation-reimbursement) are correctly recognized and managed.

While this representation closely mirrors the complete model, it cannot yet be deemed optimal due to the absence of the correct identification of the operation that initiates the process (*receive_expense_report*) and the clarification of the order of execution concerning supervisor approval and reimbursement.

Nonetheless, the final observation is encouraging, as the increase in constraint extraction instructions results in the identification of fewer constraints that are more representative of the process.

Comparing the expected results with the actual constraints extracted, we observe how in all phases the LLM extracts only the most relevant constraints rather than providing the complete list of constraints. This is highly impacted, as previously said, by the mechanism by which the LLM replies, providing the text with the highest probability rather than all possible texts.

Nonetheless, the final observation is positive since we observe that with the increase in constraint extraction instructions, fewer constraints are identified, but more representative of the process.

Tables 9 and 10, following this analysis, present the results obtained from the extraction process based on the textual description.

Approach	Extracted Constraints	Extracted Activities	Interaction
<i>Basic templates</i>	13	8	Yes
<i>Additional templates</i>	17	11	No
<i>Hierarchical Rules</i>	7	11	No

Table 9: Volume of results of extraction process of doc-9.5

Approach	Extracted Constraints	Extracted Activities
<i>Basic templates</i>	<ol style="list-style-type: none"> 1. existence(review_report) 2. existence(send_cancellation_notice) 3. response(review_report, automatic_approval) 4. response(review_report, supervisor_approval) 5. response(supervisor_approval, reimbursement) 6. response(automatic_approval, reimbursement) 7. response(send_rejection_notice, re_submit_report) 8. response(send_cancellation_notice, re_submit_report) 9. response(review_report, send_approval_in_progress_email) 10. precedence(supervisor_approval, reimbursement) 11. precedence(send_cancellation_notice, re_submit_report) 12. not-co-existence(send_rejection_notice, reimbursement) 13. not-succession(reimbursement, send_cancellation_notice) 	<ol style="list-style-type: none"> 1. send_approval_in_progress_email 2. re_submit_report 3. automatic_approval 4. review_report 5. supervisor_approval 6. send_cancellation_notice 7. reimbursement 8. send_rejection_notice
<i>Additional templates</i>	<ol style="list-style-type: none"> 1. at-most(supervisor_approval) 2. existence(send_rejection_notice) 3. existence(reimbursement) 4. existence(send_approval_in_progress_email) 5. existence(send_cancellation_notice) 6. response(expense_report_received, create_account) 7. response(rejection, send_rejection_notice) 8. response(review_report, send_approval_in_progress_email) 9. response(review_report, stop_process) 10. response(stop_process, send_cancellation_notice) 11. response(send_cancellation_notice, resubmit_expense_report) 12. not-succession(review_report, reimbursement) 13. Init(expense_report_received) 14. not-response(rejection, reimbursement) 15. not-chain-response(review_report, reimbursement) 16. succession(expense_report_received, review_report) 17. succession(rejection, send_rejection_notice) 	<ol style="list-style-type: none"> 1. send_approval_in_progress_email 2. create_account 3. rejection 4. stop_process 5. review_report 6. resubmit_expense_report 7. supervisor_approval 8. send_cancellation_notice 9. reimbursement 10. send_rejection_notice 11. expense_report_received

<i>Hierarchical Rules</i>	<ol style="list-style-type: none"> 1. existence(Reviewing report) 2. response(Rejection, Sending rejection notice) 3. response(not completed within 7 days, Sending approval in progress email) 4. precedence(Receiving Expense Report, Creating new account) 5. not-response(Rejection, Reimbursement to direct deposit) 6. not-response(not finished within 30 days, Reimbursement to direct deposit) 7. choice(Automatic approval, Approval by supervisor) 	<ol style="list-style-type: none"> 1. ReceivingExpenseReport 2. Automaticapproval 3. Sendingapprovalinprogressemail 4. notcompletedwithin7days 5. Creatingnewaccount 6. Reviewingreport 7. Reimbursementtodirectdeposit 8. notfinishedwithin30days 9. Rejection 10. Sendingrejectionnotice 11. Approvalbysupervisor
---------------------------	--	--

Table 10: Results of the extraction process of doc-9.5

4.4 - Case Study with Process 43

To offer an overview of the procedure, this document presents the description of document 10.11 extracted from the PET dataset, accompanied by a visual representation of the execution flow as illustrated in Figure 7.

The GO requests the measurements to the MSP. The MSP checks the received request. The MSP denies the request of the GO or the MSP performs the measurement. The MSP informs the GO about the failure of the reading or the MSP transmits the measured values to the GO. The GO processes the measured values. The GO sends the changed values to the MSP. The GO transmit the readings to the SP.

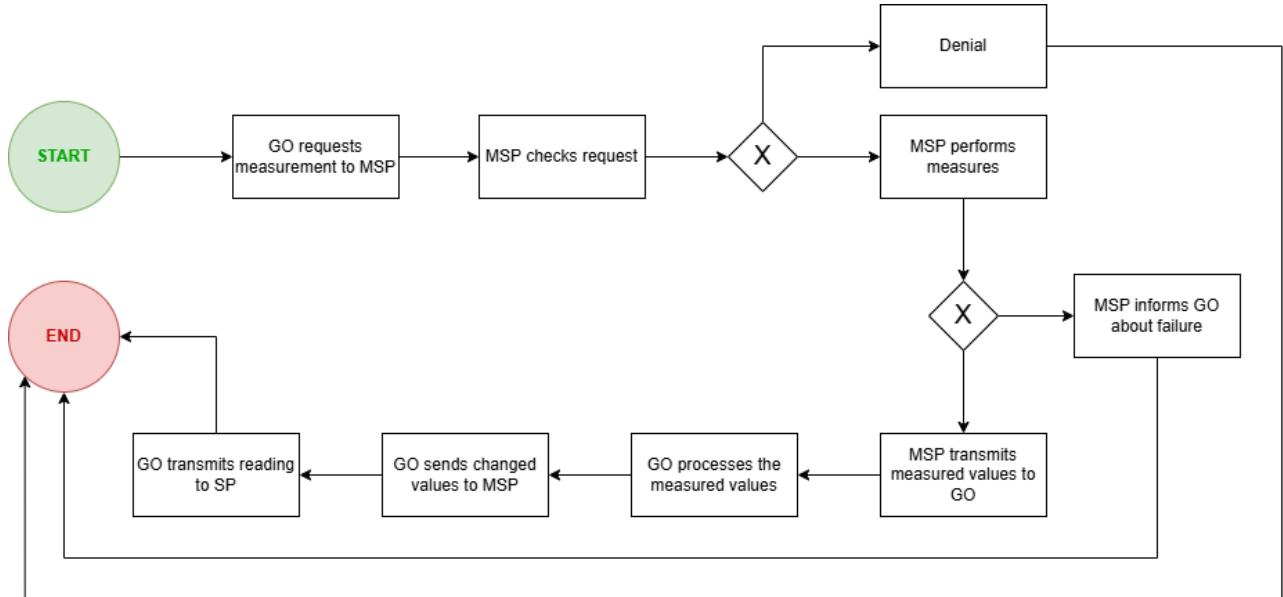


Figure 7: Visual representation of doc-10.11

In the analysis of this third and final process examined, it is noted that, similar to previous processes, the initial extraction methods correctly **recognized** seven **activities**. Even though the initial and concluding operations of the process are absent in this evaluation, they are correctly identified by the second and third methodologies, bringing the total number of recognized activities to nine, thereby acknowledging all steps of the process.

We will now share the insights derived from the analysis of the declarative constraints obtained through the three methodologies employed. We utilized the visual representation to identify the declarative constraints that should have been defined by the LLM in each approach, generating a ground truth. The results are presented in Table 11.

Basic templates	Additional templates	Hierarchical Rules
<p>1. at-most(...) for each of the 7 identified activities</p> <p>2. existence(...) for each of the 7 identified activities</p> <p>3. response(transmit_measured_values, process_measured_values)</p> <p>4. response(process_measured_values, send_changes)</p> <p>5. precedence(check_request, denial)</p> <p>6. precedence(check_request, perform_measures)</p> <p>7. precedence(perform_measures, inform_failure)</p> <p>8. precedence(perform_measures, transmit_measured_values)</p> <p>9. precedence(transmit_measured_values, process_measured_values)</p> <p>10. precedence(process_measured_values, send_changes)</p> <p>11. co-existence (transmit_measured_values, process_measured_values)</p> <p>12. co-existence (process_measured_values, send_changes)</p> <p>13. not-co-existence (denial, perform_measurements)</p> <p>14. not-co-existence (inform_failure, transmit_measured_values)</p> <p>15. not-succession(denial, perform_measurements)</p> <p>16. not-succession(inform_failure, transmit_measured_values)</p> <p>17. responded-existence(...) for each pair of activities related by response constraints</p>	<p>1. at-most(...) for each of the 9 identified activities</p> <p>2. existence(...) for each of the 9 identified activities</p> <p>3. response(request_measurements, check_request)</p> <p>4. response(transmit_measured_values, process_measured_values)</p> <p>5. response(process_measured_values, send_changes)</p> <p>6. response(send_changes, transmit_readings)</p> <p>7. precedence(request_measurements, check_request)</p> <p>8. precedence(check_request, denial)</p> <p>9. precedence(check_request, perform_measures)</p> <p>10. precedence(perform_measures, inform_failure)</p> <p>11. precedence(perform_measures, transmit_measured_values)</p> <p>12. precedence(transmit_measured_values, process_measured_values)</p> <p>13. precedence(process_measured_values, send_changes)</p> <p>14. precedence(send_changes, transmit_readings)</p> <p>15. co-existence(request_measurements, check_request)</p> <p>16. co-existence(transmit_measured_values, process_measured_values)</p> <p>17. co-existence(process_measured_values, send_changes)</p> <p>18. co-existence(send_changes, transmit_readings)</p> <p>19. not-co-existence(denial, perform_measurements)</p> <p>20. not-co-existence(inform_failure, transmit_measured_values)</p> <p>21. not-succession(denial, perform_measurements)</p> <p>22. not-succession(inform_failure, transmit_measured_values)</p>	<p>1. at-most(...) for each of the 9 identified activities</p> <p>2. existence(...) for each of the 9 identified activities</p> <p>3. Init(request_measurements)</p> <p>4. not-co-existence (denial, perform_measures)</p> <p>5. not-co-existence (inform_failure, transmit_measured_values)</p> <p>6. chain-succession(...) for each pair of activities related by succession constraints</p> <p>7. not-succession(...) for each pair of activities related by not-co-existence constraints</p> <p>8. not-response(...) for each pair of activities related by not-co-existence constraints</p> <p>9. not-precedence(denial, perform_measures)</p> <p>10. not-precedence (perform_measures, denial)</p> <p>11. not-precedence (inform_failure, transmit_measured_values)</p> <p>12. not-precedence (transmit_measured_values, inform_failure)</p> <p>13. exclusive-choice(...) for each pair of activities related by choice constraints</p> <p>14. chain-succession(...) for each pair of activities related by succession constraints</p>

	<p>23. responded-existence(...) for each pair of activities related by response constraints</p> <p>24. Init(request_measurements)</p> <p>25. not-responded-existence(...) for each pair of activities related by not-co-existence constraints</p> <p>26. not-succession(...) for each pair of activities related by not-co-existence constraints</p> <p>27. not-response(...) for each pair of activities related by not-co-existence constraints</p> <p>28. not-precedence(denial, perform_measures)</p> <p>29. not-precedence(perform_measures, denial)</p> <p>30. not-precedence(inform_failure, transmit_measured_values)</p> <p>31. not-precedence(transmit_measured_values, inform_failure)</p> <p>32. succession(...) for each pair of activities related by response and/or precedence constraints</p> <p>33. choice(denial, perform_measures)</p> <p>34. choice(inform_failure, transmit_measured_values)</p> <p>35. exclusive-choice(...) for each pair of activities related by choice constraints</p> <p>36. chain-succession(...) for each pair of activities related by succession constraints</p> <p>37. chain-response(...) for each pair of activities related by response constraints</p> <p>38. chain-precedence(...) for each pair of activities related by precedence constraints</p>	
--	--	--

Table 11: Expected declarative constraints for each approach analysis

The **basic templates** approach accurately captures the flow of the process in its entirety, effectively addressing the XOR gateways related to the approval of the measurement request and the possible failure of the measurement process.

In contrast to prior observations, we identified a transitive relationship between the *precedence* and *response* constraints regarding *check_request*, *perform_measurement*, and *transmit_measurement*.

As previously noted, the **method utilizing the additional templates** successfully identifies the beginning of the process with the *Init(request_measurement)* constraint. Nonetheless, it fails to represent the exclusive choice between the approval and denial of the measurement request.

Finally, the **hierarchical rules' method** provides a correct depiction of the process execution flow from start to finish, while also representing the junctions between possible paths. We observe a proper application of hierarchical rules, which prioritize *succession-type* constraints over *precedence-* and *response-type* constraints.

However, it is important to highlight that the *response(Deny_Request, Inform_Failure)* constraint has been inaccurately defined; the activity of *Inform_Failure* in the textual description of the process is intended to relate to the failure to read the data and not to deny the request for the data. This issue does not constitute a syntactical error but rather an interpretative oversight.

If we compare the expected constraints to those that have been extracted, we can clearly observe how, similarly to the previous two constraints, that oftentimes the LLM often prefers representing only a reduced set of constraints rather than the whole set, extracting the most relevant ones.

In conclusion, this final representation of the model can be regarded as a comprehensive and satisfactory representation of the process.

Tables 12 and 13 are the results from the extraction process, as derived from the provided textual description.

Approach	Extracted Constraints	Extracted Activities	Interaction
<i>Basic templates</i>	9	7	No
<i>Additional templates</i>	15	9	No
<i>Hierarchical Rules</i>	9	9	No

Table 12: Volume of results of the extraction process of doc-10.11

Approach	Extracted Constraints	Extracted Activities
<i>Basic templates</i>	<ol style="list-style-type: none"> 1. response(perform_measurement, transmit_values) 2. response(transmit_values, process_values) 3. response(process_values, send_changed_values) 4. precedence(check_request, deny_request) 5. precedence(check_request, perform_measurement) 6. precedence(check_request, transmit_values) 7. precedence(check_request, inform_failure) 8. not-co-existence(deny_request, perform_measurement) 9. not-succession(inform_failure, transmit_values) 	<ol style="list-style-type: none"> 1. deny_request 2. inform_failure 3. transmit_values 4. process_values 5. send_changed_values 6. check_request 7. perform_measurement

Additional templates	<ol style="list-style-type: none"> 1. response(check_request, deny_request) 2. response(check_request, perform_measurement) 3. response(transmit_measured_values, process_measured_values) 4. response(process_measured_values, send_changed_values) 5. response(send_changed_values, transmit_readings) 6. precedence(check_request, deny_request) 7. precedence(check_request, perform_measurement) 8. Init(request_measurement) 9. succession(request_measurement, check_request) 10. succession(perform_measurement, inform_failure) 11. succession(perform_measurement, transmit_measured_values) 12. succession(transmit_measured_values, process_measured_values) 13. succession(process_measured_values, send_changed_values) 14. succession(send_changed_values, transmit_readings) 15. exclusive-choice(inform_failure, transmit_measured_values) 	<ol style="list-style-type: none"> 1. transmit_measured_values 2. deny_request 3. request_measurement 4. inform_failure 5. process_measured_values 6. send_changed_values 7. check_request 8. transmit_readings 9. perform_measurement
Hierarchical Rules	<ol style="list-style-type: none"> 1. response(Deny_Request, Inform_Failure) 2. response(Perform_Measurement, Transmit_Measured_Values) 3. precedence(Request_Measurements, Check_Request) 4. precedence(Transmit_Measured_Values, Process_Measured_Values) 5. precedence(Process_Measured_Values, Send_Changed_Values) 6. precedence(Send_Changed_Values, Transmit_Readings) 7. not-succession(Inform_Failure, Process_Measured_Values) 8. Init(Request_Measurements) 9. exclusive-choice(Deny_Request, Perform_Measurement) 	<ol style="list-style-type: none"> 1. Deny_Request 2. Perform_Measurement 3. Process_Measured_Values 4. Transmit_Measured_Values 5. Request_Measurements 6. Check_Request 7. Send_Changed_Values 8. Inform_Failure 9. Transmit_Readings

Table 13: Results of the extraction process of doc-10.11

4.5 - Evaluation

Upon examining the results derived from the analyses of the three process descriptions, we can draw a few conclusions.

As expected, the LLM demonstrates a progressive enhancement in the extraction of accurate representations of the process model as additional information is provided. This advancement provides a solid foundation for potential future developments, for which we have outlined some possibilities in the next section.

A comparison of the constraints identified across the three cases, extracted using the three methods, reveals that certain elements within the textual descriptions are often ambiguous and lack detail. This ambiguity can lead to confusion, prompting the LLM to formulate its own interpretations.

Most of the time, this results in a correct depiction of the process execution flow; there are occasional instances, albeit rare, where the representation is incorrect.

This challenge may be mitigated and even solved by increasing the user interaction with the model; however, in this case study, we have intentionally limited such interactions to evaluate the LLM's capabilities without external influences.

Furthermore, we observed that the execution flows of these processes, similar to those of most processes found within the PET Dataset, present a linear flow with rare decision points. Loops, splits in execution flow, or other complex features that increase the complexity of the process model are absent. Consequently, we were unable to assess the LLM's performance in recognizing these elements, and regarding the application of more intricate constraints, such as *alternates*, *choices*, *chains*, and others.

The three processes analyzed in this case study offer valuable insights into the strengths of using an LLM during the discovery phase of a process. Notably, the LLM is capable of extracting a multitude of declarative constraints in a significantly shorter timeframe than would be possible without AI intervention. This is particularly beneficial when analyzing complex and intricate processes that would otherwise require extensive processing time by a domain expert.

In the analysis of the processes examined through these case studies, the comparison of the identified constraints with our predictions reveals that while the interpretations offered by the LLM are generally correct, they often fail to fully capture the anticipated constraints. This limitation arises from the LLM's tendency to present a restricted set of results, specifically those identified as most likely to be pertinent. To enhance the accuracy of our results, it may be beneficial to provide prompts with additional details regarding the volume of activities and constraints that are expected.

Conversely, these analyses also highlight certain limitations, chief among them being the requirement for human interaction. While explicit human input is not always mandatory, it becomes essential following the initial definition of constraints to clarify aspects that may have been misinterpreted by the model.

5 - Conclusion

To conclude this project, focused on the examination of the role, potential, and performance of LLMs implementation within the discovery phase of business processes, we outlined the strengths and weaknesses of the project and suggesting avenues for the future development of this research.

5.1 - Observations on LLM Performance and Results

Concluding our analysis of the research project focused on the application of Large Language Models in the discovery phase of declarative processes, we shall summarize the most relevant strengths and limitations identified throughout our investigation.

Among the strengths of this approach, it is necessary to highlight the significant efficiency gains that LLMs offer in terms of **timing** during the extraction phase. Specifically, generative AI can effectively extract constraints from extensive texts within a few tens of seconds, immensely reducing the time required compared to manual extraction processes, which would inevitably demand a considerably longer duration.

Moreover, the capability to derive constraints directly from a textual description of the process **eliminates the necessity for supplementary documentation**. This stands in contrast to traditional mining algorithms that require complete representations of process executions in the form of event logs before being able to formulate declarative constraints.

The integration of generative AI enhances the extraction process possibilities to interpret **unstructured data** that otherwise would be unusable for analysis performed through the use of traditional process mining approaches.

Lastly, but certainly not least, an important advantage of this methodology lies in the accessibility of **open-source models**, which facilitate the implementation of this analysis approach and allow the fine-tuning of the LLM. These resources allow user-model interaction through the usage of an API key, granting the possibility of fine-tuning the model without necessitating the investment of capital to develop ad-hoc models.

Addressing the limitations of this approach, we must first highlight the main limits of generative AI: its inability to comprehend the input it receives, operating through pattern recognition utilizing probability distribution functions to predict the outcome. This restriction can sometimes lead to incorrect interpretations or even to **hallucination**, wherein plausible yet false information is generated.

Furthermore, the models that offer free API keys typically represent less advanced versions or rely on limited datasets to predict results, often using a limited set of parameters. Additionally, these models frequently impose **low limits** on the number of **tokens** per time unit and request. While this may not represent a significant issue for simpler process descriptions, it can cause challenges regarding scalability in more complex systems.

Lastly, another notable limitation of this project, as previously discussed, is the **lack of a ground truth** that could facilitate a quantitative assessment of the performance of the LLM. Indeed, conducting an analysis focused on precision and recall of the outputs would strongly enhance the accuracy of the LLM performance evaluation.

This challenge is caused by the fact that this remains an underdeveloped research field, with a limited availability of datasets offering annotated process descriptions. Furthermore, to our knowledge, there are currently no datasets available offering annotated descriptions containing the necessary constraints to accurately represent the declarative model of the process.

5.2 - Further Possible Development

Considering the potential directions for future research development, it is necessary to begin observing those developments that would solve the issues identified as the research limitations.

To mitigate possible misinterpretations and inaccuracies generated by the LLM, increased **user interaction** would be beneficial. This could involve prompting the model to express its "Chain-of-Thought", allowing users to better understand and validate the constraints generated.

With the availability of funding, there is an opportunity to leverage **paid open-source** models. Such models typically offer enhanced performance due to their access to larger training data pools and an increased number of parameters, resulting in more precise and pertinent responses.

Furthermore, it would be interesting to propose and implement the application of these methodologies in tools designed for representing and analyzing declarative models, similarly to Delare4Py, in programming languages like Java, which have been less used in machine learning and process mining compared to Python.

Given the good performance observed in this research, it may also be feasible to explore the application of LLMs in the development of **procedural models** deriving structures like Petri Nets from unstructured textual input.

In considering future implementations of this research, it would be beneficial to **increase the constraint templates** provided to the LLM.

Furthermore, it would be interesting to integrate the redundancy types presented in "A knowledge-based integrated approach for discovering and repairing declare maps." (Maggi, 2013) in a new prompt. By providing more instruction it could facilitate the **elimination** of various forms of **redundancy** beyond the implicational redundancy already addressed through hierarchical rules. For instance, addressing derivable redundancies, syntactic redundancies, and the representation of non-influential constraints would significantly improve the readability, facilitating the understanding of the process and, at the same time, minimizing potential logical errors during result verification.

Moreover, a stimulating development for improvement can be identified in the integration of **LLMs** within the scope of **image analysis**. Users could provide a representative image of a flowchart that illustrates the process execution as a whole, which would be used by the LLM to extract declarative constraints and represent the declarative model. This would expand significantly the already wide range of possible inputs to be used in the automatic extraction of declarative constraints.

Undoubtedly, there exist several potential developments in this domain; however, we aim to present those we believe are most pertinent to advancing research and enhancing its effectiveness across diverse application areas.

Bibliography

Ackermann, Lars, Julian Neuberger, and Stefan Jablonski. "Data-driven annotation of textual process descriptions based on formal meaning representations." *International Conference on Advanced Information Systems Engineering*. Cham: Springer International Publishing, 2021.

Bellan, Patrizio, et al. "Process extraction from text: benchmarking the state of the art and paving the way for future challenges." *arXiv preprint arXiv:2110.03754*, 2021.

Bellan, Patrizio. *Process Extraction from Text: a Reference Corpus, a Benchmark, and a Generative AI Approach*. PhD Dissertation Free University of Bozen-Bolzano, 2024.

Berti, Alessandro, Humam Kourani, and Wil MP van der Aalst. "PM-LLM-Benchmark: Evaluating large language models on process mining tasks." *International Conference on Process Mining*. Cham: Springer Nature Switzerland, 2024.

Donadello, Ivan, et al. "Declare4Py: A Python Library for Declarative Process Mining." *BPM (PhD/Demos)* 3216, 2022: 117-121.

Donadello, Ivan, et al. "Outcome-oriented prescriptive process monitoring based on temporal logic patterns." *Engineering Applications of Artificial Intelligence* 126 (2023): 106899.

Du, Weihong, et al. "PAGED: A Benchmark for Procedural Graphs Extraction from Documents." *arXiv preprint arXiv:2408.03630*, 2024.

Dumas, Marlon, et al. *Fundamentals of business process management*. Vol. 1. Heidelberg: Springer 2013.

Friedrich, Fabian, Jan Mendling, and Frank Puhlmann. "Process model generation from natural language text." *Advanced Information Systems Engineering: 23rd International Conference, CAiSE 2011, London, UK, June 20-24, 2011. Proceedings* 23. Springer Berlin Heidelberg, 2011.

Guide, B.A.B.O.K. "2.0: A Guide to the Business Analysis Body of Knowledge." *International Institute of Business Analysis. IIBA*, 2012.

Maggi, Fabrizio M., RP Jagadeesh Chandra Bose, and Wil MP van der Aalst. "A knowledge-based integrated approach for discovering and repairing declare maps." *Advanced Information Systems Engineering: 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013. Proceedings* 25. Springer Berlin Heidelberg, 2013.

Norouzifar, Ali, et al. "Bridging domain knowledge and process discovery using large language models." *International Conference on Business Process Management*. Cham: Springer Nature Switzerland, 2024.

Pesic, Maja, Helen Schonenberg, and Wil MP Van der Aalst. "Declare: Full support for loosely-structured processes." *11th IEEE international enterprise distributed object computing conference (EDOC 2007)*. IEEE, 2007.

Qian, Chen, et al. "An approach for process model extraction by multi-grained text classification." *Advanced Information Systems Engineering: 32nd International Conference, CAiSE 2020, Grenoble, France, June 8–12, 2020, Proceedings 32*. Springer International Publishing, 2020.

Radford, Alec, et al. "Improving language understanding by generative pre-training.", 2018.

Schunselaar, Dennis MM, Fabrizio M. Maggi, and Natalia Sidorova. "Patterns for a log-based strengthening of declarative compliance models." *International Conference on Integrated Formal Methods*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

Van Der Aalst, Wil MP. "Bridging Domain Knowledge and Process Discovery Using Large Language Models." *Business Process Management Workshops*, 2024.

Van Der Aalst, Wil, et al. "Process mining manifesto." *Business Process Management Workshops: BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I 9*. Springer Berlin Heidelberg, 2012.

Van Der Aalst, Wil. "Process mining: Overview and opportunities." *ACM Transactions on Management Information Systems (TMIS) 3.2*, 2012: 1-17.

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems 30* , 2017.

Appendix

Prompts

The following are the complete versions of the templates used in the different approaches of the process descriptions analysis.

Basic Templates

Process discovery is a type of process mining aiming to represent the process model explaining how the process is executed. Our focus is on the discovery of an imperative process model. Recently, we developed a framework that allows for some declarative constraints as input to discover better imperative models. Declarative models are close to natural language, however, for a domain expert, they might not be familiar.

Therefore, we need your help to translate the process description given as text to the declarative constraints we need for the discovery of imperative process models.

Consider only the following declarative constraint definitions where a, b, c, and d are activities and template example-template(x,y) specifies a template named "example-template" and x and y can be one of the activities from the set of all activities. Do not reason based on the template names and only use explanations to understand the meaning of specific templates:

1. at-most(A): A occurs at most once.
 - a. Some examples satisfying this constraint:
 - i. The process starts with activity B for setup, followed by A for approval, then C for data collection, and ends with D for implementation. Optional execution of B can occur again before C.
 - ii. The workflow begins with C for initial data collection, followed by D for setup, then A for mandatory approval, and concludes with B for detailed planning.
 - iii. The process initiates with D for setup, followed by B for planning, C for data analysis, and finally A for essential approval. Optional execution of C may occur again after B.
 - b. Some examples violating this constraint:
 - i. The process starts with A for initial approval, followed by B for planning, then loops back to A for re-approval, and ends with C for data analysis and D for implementation.
 - ii. Begin with C for data collection, followed by A for approval, then B for planning, another A for re-approval, and ends with D for execution.
 - iii. The workflow involves D for setup, followed by parallel execution of A for initial approval and B for planning, then A second occurrence of A for final approval, and C for final data analysis.
2. existence(A): A occurs at least once, which means the existence of activity A is mandatory.
 - a. Some examples satisfying this constraint:
 - i. The process starts with activity B for initial setup, followed by A for mandatory approval, then C for data collection, and finally D for implementation. Optional execution of C can occur again before d.
 - ii. The workflow begins with D for setup, followed by A for critical approval, then parallel execution of B for planning and C for analysis, and concludes with D for finalization.
 - iii. Activity C starts with data gathering, then loops to B for planning, followed by A for essential approval, and ends with D for implementation.
 - iv. all the customers start the process with activity A.
 - b. Some examples violating this constraint:
 - i. The process starts with B for initial tasks, followed by C for data analysis, and concludes with D for implementation. Optional execution of C can occur again before D.
 - ii. Begin with D for setup, followed by B for detailed planning, then C for further analysis, and ends with D for final execution.
 - iii. The workflow involves C for initial data collection, followed by parallel execution of B for planning and D for implementation, and ends with another round of C.
 - iv. Either one of the activities A or B occurs. Each trace can only have A or B.
3. response(A,B): If A occurs, then B occurs after A.
 - a. Some examples satisfying this constraint:
 - i. The process starts with C for data collection, then A for approval, followed by B for planning, and finally D for implementation. Optional execution of C may occur again before D.

- ii. Activity D begins with preliminary tasks, then A for approval, followed by parallel execution of B for planning and C for analysis, ending with D for finalization.
 - iii. The workflow initiates with A for approval, loops back to A second approval if needed, then proceeds to B for detailed planning, and concludes with C for execution.
 - iv. All the occurrences of activity A should be followed by activity B.
 - b. Some examples violating this constraint:
 - i. The process starts with B for initial setup, followed by C for data analysis, then A for approval, and ends with D for implementation.
 - ii. Begin with C for data gathering, followed by A for approval, then D for execution, and optionally B for post-implementation review.
 - iii. The process involves D for setup, followed by A for approval, and then C for analysis. An optional activity B for final planning may or may not occur.
4. precedence(A,B): B occurs only if preceded by A.
- a. Some examples satisfying this constraint:
 - i. The process begins with activity A for approval, followed by C for data collection, and then B for planning. Optional execution of D can occur anytime before or after B.
 - ii. Activity D initiates the setup, then A for mandatory approval, followed by parallel execution of C for analysis and B for planning, and concludes with D for finalization.
 - iii. The workflow starts with C for initial tasks, followed by A for approval, then B for detailed planning, and ends with A loop back to C for further analysis if needed.
 - iv. After execution of activity A, some cases will continue with activity B and some cases will continue with activity C.
 - b. Some examples violating this constraint:
 - i. The process starts with C for data gathering, followed by B for planning, then A for approval, and ends with D for implementation.
 - ii. Begin with D for initial setup, followed by parallel execution of B for planning and C for analysis, and finally A for approval.
 - iii. The workflow involves D for setup, followed by C for data collection, B for planning, and ends with an optional activity A for final approval.
5. co-existence(A,B): A and B occur together.
- a. Some examples satisfying this constraint:
 - i. The process starts with B for initial setup, followed by A for approval, then C for data collection, and ends with D for implementation. Optional execution of B can occur again before C.
 - ii. Activity A begins with approval, followed by parallel execution of B for planning and C for data analysis, and concludes with D for finalization.
 - iii. The workflow starts with D for setup, followed by A for approval, B for planning, and ends with C for execution. Optional execution of C may occur again after B.
 - b. Some examples violating this constraint:
 - i. The process begins with C for data collection, followed by D for setup, then B for planning, and ends with another round of C.
 - ii. Start with D for setup, followed by C for data analysis, then A for approval, and ends with C for final review.
 - iii. The workflow involves C for initial data gathering, followed by parallel execution of D for setup and B for planning, and concludes with C for finalization.
6. not-co-existence(A,B): A and B never occur together.
- a. Some examples satisfying this constraint:
 - i. The process starts with A for approval, followed by C for data collection, then D for implementation. Optional execution of C can occur again before D.
 - ii. Begin with B for initial setup, followed by C for analysis, then D for execution. An optional activity C can repeat after D.
 - iii. The workflow starts with C for data gathering, then moves to D for setup, followed by optional repetitions of C.
 - iv. In our process activities A and B cannot occur together.
 - v. Some cases have activity A and the other cases have activity B.
 - vi. If activity A does not occur, then activity B will occur.
 - b. Some examples violating this constraint:
 - i. The process begins with A for approval, followed by B for planning, then C for data collection, and ends with D for implementation.
 - ii. Start with C for data analysis, then A for approval, followed by D for execution, and ends with B for final review.

- iii. The workflow involves D for setup, followed by parallel execution of A for approval and B for planning, and concludes with C for finalization.
- 7. not-succession(A,B): B cannot occur after A.
 - a. Some examples satisfying this constraint:
 - i. The process starts with A for approval, followed by C for data collection, and ends with D for implementation. Optional execution of C can occur again before D.
 - ii. Begin with B for initial setup, followed by D for setup, then A for approval, and finally C for analysis.
 - iii. The workflow starts with C for data gathering, followed by A for approval, then D for execution, and optional repetitions of C.
 - b. Some examples violating this constraint:
 - i. The process begins with A for approval, followed by B for planning, then C for data collection, and ends with D for implementation.
 - ii. Start with C for data analysis, then A for approval, followed by D for execution, and ends with B for final review.
 - iii. The workflow involves D for setup, followed by parallel execution of A for approval and C for analysis, then B for planning, and concludes with C for finalization.
- 8. responded-existence(A,B): If A occurs in the trace, then B occurs as well.
 - a. Some examples satisfying this constraint:
 - i. The process starts with B for initial setup, followed by A for approval, then C for data collection, and ends with D for implementation. Optional execution of B can occur again before c.
 - ii. Begin with A for initial approval, followed by D for setup, then B for planning, and finally C for analysis.
 - iii. The workflow starts with C for data gathering, followed by A for approval, then D for execution, and optional repetitions of B for planning.
 - b. Some examples violating this constraint:
 - i. The process begins with A for approval, followed by C for data collection, then D for implementation. Optional execution of C can occur again before d.
 - ii. Start with C for data analysis, followed by A for approval, then D for execution, and ends with C for final review.
 - iii. The workflow involves D for setup, followed by A for approval, then parallel execution of C for analysis and B for planning, and concludes with C for finalization.

Some more instructions:

- It is not possible to generate constraints like response(a, (b or c)). The first and second elements must be a single activity

For each task, I provide the set of activity labels that exist in the process with a short description. Then, I present a text written by a process expert and want you to translate it to declarative constraints and write it in a plaintext block.

Additional Templates

In addition to the constraint models provided above, there are others:

1. alternate-succession(A,B): Between any two A events, there must be exactly one B. So, B must occur after each A, but no two As without a B in between. A and B alternate.
 - a. Some examples satisfying this constraint:
 - i. Each request (A) must be followed by a decision (B), and you can't make another request until a decision is made.
 - ii. The assistant submits a report (A), which is immediately approved by the manager (B), and then another report is submitted (A) and approved (B).
 - iii. After a submission (A), there's a timely approval (B), followed by archiving.
 - iv. The process involves submitting (A) → approval (B) → submission (A) → approval (B), with no skipped approvals.
 - b. Some examples violating this constraint:
 - i. Two submissions (A) occur in a row before any approval (B) is given.
 - ii. A submission (A) happens, then a discussion takes place, and another submission (A) occurs without any approval (B) in between.
 - iii. A report is submitted (A), approved (B), then submitted again (A), but no second approval (B) follows.
2. Init(a): A must be the first event in every process of execution. The process starts with A.
 - a. Some examples satisfying this constraint:
 - i. A login (A) must be the first action in a session.
 - ii. The user first starts a session (A), then proceeds to log in and access the dashboard.
 - iii. The workflow begins when the system starts a session (A) before allowing the user to browse products.
 - iv. A session is initiated (A), followed by profile viewing and editing.
 - b. Some examples violating this constraint:
 - i. The user logs in (B) before the session is started (A).
 - ii. Dashboard access occurs first (B), and the session start (A) is recorded later.
 - iii. The flow begins with product browsing, not with session start (A) as required.
3. not-responded-existence(A,B): If A occurs, then B must not occur anywhere in the process. The presence of A excludes B.
 - a. Some examples satisfying this constraint:
 - i. If an emergency shutdown (A) occurs, then no restart (B) can happen in the same trace.
 - ii. The fraud detection tool flags the account (A), and the team proceeds to suspend it without unlocking (B).
 - iii. After an audit, the system flags an account (A), and a report is generated - no unlocking (B) happens.
 - iv. The agent flags the account (A), which leads directly to closure (C), bypassing any unlock (B).
 - b. Some examples violating this constraint:
 - i. The system flags the account (A) and then later unlocks it (B) after user verification.
 - ii. Following a flagging (A), the account is temporarily unlocked (B) to retrieve data.
 - iii. The sequence B → A → C violates the constraint because both A and B happen within the same process.
 - iv. The account is flagged (A) during review, and later unlocked (B) for continued use.
4. not-response(A,B): If A happens, then B must not happen afterward. B is not allowed after A.
 - a. Some examples satisfying this constraint:
 - i. If an invoice is canceled (A), no payment (B) should happen after.
 - ii. The technician closes the ticket (A) and archives it without any further action.
 - iii. After closing the ticket (A), the system logs the case as resolved with no reopening (B).
 - iv. A ticket is closed (A) and followed by a feedback request, with no reopen (B) in the workflow.
 - v. The sequence B ◊ A ◊ C is allowed
 - b. Some examples violating this constraint:
 - i. The ticket is closed (A) and then later reopened (B) due to a missed issue.
 - ii. Closing (A) is followed by a user complaint, leading to a reopen (B).
 - iii. After a ticket is closed (A), a second-level agent reopens it (B) for escalation.

5. **not-precedence(A,B):** If B occurs, then A must not have occurred before. A must not precede B.
- Some examples satisfying this constraint:
 - If a refund is issued (B), it must not be preceded by a delivery (A).
 - The system issues a refund (B) automatically based on transaction failure, without any complaint submission (A) from the user.
 - A refund (B) is provided proactively during a recall process, with no customer complaints (A) involved.
 - The support team issues a refund (B) before any chance of submitting a complaint (A) by the user.
 - Some examples violating this constraint:
 - The customer submits a complaint (A), and then the company issues a refund (B) in response.
 - After several complaint submissions (A), a refund (B) is processed.
 - A ticket involving a complaint (A) leads directly to a refund (B) approval.
6. **not-chain-response(A,B):** It is forbidden that B occurs immediately after A. B must not directly follow A.
- Some examples satisfying this constraint:
 - After submitting a form (A), an approval (B) must not happen immediately.
 - The user saves a draft (A), reviews the data (C), and then submits the form (B).
 - After saving the draft (A), the document is edited (C) and only then submitted (B).
 - The process includes draft saving (A), a peer review, and finally form submission (B).
 - Some examples violating this constraint:
 - The user saves the draft (A) and immediately submits the form (B).
 - Save draft (A) is followed directly by submission (B) without any checks.
 - A and B occur consecutively with no intermediate validation step.
7. **not-chain-precedence(A,B):** If B occurs, it must not have been immediately preceded by A. A must not be directly before B.
- Some examples satisfying this constraint:
 - If a task is marked complete (B), it must not come right after a comment (A).
 - The system checks account balance (C), then grants access (B) - identity validation (A) was done earlier or skipped.
 - After completing a background check (C), the system grants access (B), not immediately after identity validation (A).
 - Access is granted (B) following a security questionnaire (C), with no direct preceding validation (A).
 - Some examples violating this constraint:
 - The system validates identity (A) and immediately grants access (B).
 - A user passes identity check (A) and access is granted (B) right after, violating the constraint.
8. **succession(A,B):** If A occurs, then B must also occur (at some point later), and vice versa: if B occurs, then A must have occurred before. A and B are connected; one implies the other, in order.
- Some examples satisfying this constraint:
 - If a customer places an order (A), then an invoice must be issued (B) and if an invoice is issued, the order must have been placed
 - The process begins with (A) requesting creation, followed by (B) requesting approval, then proceeding to C and D for execution.
 - Workflow starts with C, followed by A for initial check, then d, and finally B for final approval.
 - Some examples violating this constraint:
 - The process starts with A, then goes to C and D, but B is never executed.
 - B occurs early in the process, followed by C, but A never takes place beforehand.
 - The workflow has A at the beginning, C in the middle, and ends with D - B is completely missing.
9. **choice(A,B):** A or B have to occur at least once.
- Some examples satisfying this constraint:
 - The process begins with A for document review, followed by C, and ends with D. B never occurs.
 - The workflow includes B for data approval after C, but A is not present.
 - The trace includes both A and B, with A occurring before C, and B happening after D.
 - Some examples violating this constraint:
 - The trace includes only C, D, and E - neither A nor B are present.

- ii. Workflow starts with C, then moves to D, and ends with E. No trace of A or B.
 - iii. The entire process involves only repeated executions of C and D, excluding both A and B.
10. exclusive-choice(A,B): A or B have to occur at least once but not both.
- a. Some examples satisfying this constraint:
 - i. The process starts with A for submission, continues with C, and ends with D - B never appears.
 - ii. Only B for manual override occurs midway through the trace; A is never executed.
 - iii. The process starts with C then reaches a step where a condition is either verified or not, based on the condition result the process proceeds with A or B, only one is chosen
 - b. Some examples violating this constraint:
 - i. The process includes A for creation, followed by B for confirmation later in the trace.
 - ii. After C, both A and B are executed as part of a dual-check process.
 - iii. The workflow loops through both A and B multiple times before finalizing with D.
11. not-exclusive-choice(A,B): It is not allowed that only one of A or B occurs; if one happens, the other must also occur. Both A and B must either happen together or not at all.
- a. Some examples satisfying this constraint:
 - i. If A contract is signed (A), then onboarding (B) must also occur, and vice versa.
 - ii. The process includes A for creation, followed by B for confirmation later in the trace.
 - iii. After C, both A and B are executed as part of a dual-check process.
 - iv. The workflow loops through both A and B multiple times before finalizing with D.
 - b. Some examples violating this constraint:
 - i. Only A is executed in the trace, without A corresponding B.
 - ii. The workflow contains B for approval but skips A entirely.
 - iii. A appears early in the process, but B never occurs, violating the required pairing.
12. not-chain-succession(A,B): It is forbidden that B directly follows A (i.e., with no events in between). B can happen after A, but not immediately after.
- a. Some examples satisfying this constraint:
 - i. A system check (B) must not follow a reboot (A) immediately.
 - ii. The process starts with A, followed by C, and only then B for validation.
 - iii. A initiates the workflow, followed by D, E, and then B appears toward the end.
 - iv. B appears early, then A occurs, and later C finishes the trace. At no point does B directly follow A.
 - b. Some examples violating this constraint:
 - i. The trace begins with A, and B occurs immediately after as the second event.
 - ii. A is executed, followed directly by B, with no intermediary steps.
 - iii. A mid-process A is followed instantly by B without interruption.
13. chain-succession(A,B): A and B occur in the process instance if and only if the latter immediately follows the former.
- a. Some examples satisfying this constraint:
 - i. After scanning a product (A), the system must immediately log the scan result (B).
 - ii. The trace is A → B → C → D. Every A is immediately followed by B, and every B has A just before it.
 - iii. The process includes C, then A → B, and then D.
 - iv. A single A → B pair appears, with no other instances of A or B elsewhere in the trace.
 - v. A sequence like C → A → B → A → B → C is allowed by this constraint
 - b. Some examples violating this constraint:
 - i. A is followed by C, with B appearing later - B does not directly follow A.
 - ii. B appears without A preceding A.
 - iii. One A is followed by B, but another B shows up without any prior A.
 - iv. Sequences like B → C → A → A → C and B → C → A → A → B → C are not allowed.
14. chain-response(A,B): Each time A occurs in the process instance, then B occurs immediately afterwards, with no other action in between.
- a. Some examples satisfying this constraint:
 - i. A → B → C → D: every A is directly followed by B.
 - ii. C → B → D: B can occur even without A prior.
 - iii. The trace has one A → B sequence; any B not preceded by A is still allowed.
 - b. Some examples violating this constraint:
 - i. A → C → B: B does not immediately follow A.
 - ii. A appears but is followed by D, not B.
 - iii. Multiple A events occur, but none are immediately followed by B.

15. chain-precedence(A,B): Each time B occurs in the process instance, then A occurs immediately beforehand.

a. Some examples satisfying this constraint:

- i. $A \rightarrow B \rightarrow C \rightarrow D$: B occurs immediately after A and nowhere else.
- ii. $C \rightarrow A \rightarrow B \rightarrow D$: B is always and only right after A.
- iii. $A \rightarrow C \rightarrow D$: A can occur even without B

b. Some examples violating this constraint:

- i. B occurs first without any A before it.
- ii. $A \rightarrow C \rightarrow B$: B is not immediately after A.
- iii. B appears multiple times, but not every instance is preceded by A directly.

16. alternate-response(A,B): If A occurs, then B must eventually follow without any other A in between.

a. Some examples satisfying this constraint:

- i. $A \rightarrow B \rightarrow A \rightarrow B \rightarrow C$: every A is followed by B before another A occurs.
- ii. $C \rightarrow A \rightarrow B \rightarrow D$: only one A, and it's followed by B.
- iii. $A \rightarrow B \rightarrow C \rightarrow D$: A single instance of A followed by B, with no second A.

b. Some examples violating this constraint:

- i. $A \rightarrow A \rightarrow B$: second A occurs before B appears.
- ii. $A \rightarrow C \rightarrow A \rightarrow B$: B doesn't follow the first A before the second A.
- iii. $A \rightarrow D \rightarrow A \rightarrow C \rightarrow B$: multiple As without Bs in between.

17. alternate-precedence(A,B): B can occur only if A has occurred before, without any other B in between.

a. Some examples satisfying this constraint:

- i. The employee submits the report (A) before the manager approves it (B), and then the document is finalized.
- ii. The analyst submits one report (A) and gets it approved (B), then submits another (A) which is also approved (B).

b. Some examples violating this constraint:

- i. The manager approves the report (B) even though it was never submitted (A) beforehand.
- ii. The employee submits a report (A), gets it approved (B), and then a second approval (B) happens without a new submission (A).
- iii. The system logs an approval (B) first, then a submission (A) - violating the required order.

Hierarchical relations

These constraints are related together through hierarchical relationships. If a constraint implies one or more other constraints only the stronger (highest level) should be present in the final constraint list, so if for example in the extracted list of constraints appear A, B and C but A implies B and B implies C then only A should be present in the final version of the constraints.

Here follows all the implications:

- Chain-Succession implies Chain-Response
- Chain-Response implies Alternate-Response
- Alternate-Response implies Response
- Response implies Responded-Existence
- Chain-Succession implies Chain-Precedence
- Chain-Precedence implies Alternate-Precedence
- Alternate-Precedence implies Precedence
- Chain-Succession implies Alternate-Succession
- Alternate-Succession implies Succession
- Succession implies Co-Existence
- Co-Existence implies Responded-Existence
- Succession implies Response
- Succession implies Precedence
- Alternate-Succession implies Alternate-Response
- Alternate-Succession implies Alternate-Precedence
- Init implies Precedence
- Exclusive-Choice implies Choice
- Not-Exclusive-Choice implies Not-Succession
- Not-Succession implies Not-Chain-Succession

Result Formatting Rules

The activities should not be shortened to a letter but included in the final constraint in a simplified version, such as `executed_payment`, `approval_request`, `confirmation`, ...

In our final interaction the constraint should be defined as "Final Formal Declarative Constraints: " (this line MUST appear only once in your reply, if you have any intermediate interpretations you can use "Temporary Declarative Constraints" or similar phrases) followed by one constraint per line defined using the template provided in the previous instruction.

Furthermore, in a line, you have to summarize all the activities present in the constraints, writing it as "Activities: " and on the same line, all the activities divided by commas, you cannot put a dot at the end of the line

Be succinct and do not hallucinate.

Consider this text and, if you find it necessary, ask me questions to clarify whatever may be unclear, and then extract highly reliable declarative constraints. If present, your questions should be placed at the end of your response.

Here is the process description:

{textual_description}

Case studies: LLM-User Interactions

In the following sections we report the entire user-LLM interaction resulting from the analysis of each process with the three extraction methods

Approach 1

Process 15

💻 AI: Based on the provided process description, I have identified the following activities:

- * update_group_calendars
- * create_meeting_folders
- * upload_documents
- * advise_committee_members
- * hold_meeting
- * produce_minutes
- * conduct_qa_check
- * send_minutes
- * update_departmental_records

The process description can be translated into the following declarative constraints:

Temporary Declarative Constraints:

- existence(hold_meeting)
- response(update_group_calendars, create_meeting_folders)
- response(create_meeting_folders, upload_documents)
- response(hold_meeting, produce_minutes)
- response(produce_minutes, conduct_qa_check)
- response(conduct_qa_check, send_minutes)
- response(send_minutes, update_departmental_records)
- precedence(conduct_qa_check, send_minutes)

However, after re-evaluating the description, I realized that some of these constraints can be relaxed or are implicit. Here are the refined constraints:

Final Formal Declarative Constraints:

- existence(hold_meeting)
- response(update_group_calendars, create_meeting_folders)
- response(hold_meeting, produce_minutes)

```
response(produce_minutes, conduct_qa_check)
response(conduct_qa_check, send_minutes)
precedence(conduct_qa_check, send_minutes)
at-most(conduct_qa_check)
```

Activities: update_group_calendars, create_meeting_folders, upload_documents, advise_committee_members, hold_meeting, produce_minutes, conduct_qa_check, send_minutes, update_departmental_records

Please let me know if these constraints accurately reflect the process description.

Also, I assumed that "within 5 working days" implies that there is a time constraint, but since the template provided does not support time constraints, I focused on the activity sequence and existence. If you need to incorporate time constraints, please provide the corresponding template.

Lastly, I did not include "upload_documents", "advise_committee_members", and "update_departmental_records" in the constraints as their relationships with other activities seem optional or implicit. If you could clarify their roles, I might be able to provide more accurate constraints.

Process 35

LLM 1

AI: Based on the process description, I have identified the following activities:

- * create_account
- * review_report
- * automatic_approval
- * supervisor_approval
- * send_rejection_notice
- * send_approval_in_progress_email
- * send_cancellation_notice
- * reimbursement
- * re_submit_report

Here are my temporary declarative constraints:

- * existence(review_report)
- * response(review_report, automatic_approval)
- * response(review_report, supervisor_approval)
- * precedence(supervisor_approval, reimbursement)
- * response(supervisor_approval, reimbursement)
- * response(automatic_approval, reimbursement)
- * not-co-existence(send_rejection_notice, reimbursement)
- * response(send_rejection_notice, re_submit_report)
- * at-most(send_approval_in_progress_email)
- * existence(send_cancellation_notice)
- * response(send_cancellation_notice, re_submit_report)
- * not-succession(reimbursement, send_cancellation_notice)

Please let me know if these constraints align with your expectations or if I need to make any adjustments.

Activities: create_account, review_report, automatic_approval, supervisor_approval, send_rejection_notice, send_approval_in_progress_email, send_cancellation_notice, reimbursement, re_submit_report

I will wait for your feedback before providing the final formal declarative constraints.

Also, I have a question: are "create_account" and "re_submit_report" part of the main process flow or are they optional? The text mentions that a new account must be created if the employee does not already have one, but it's not clear if this is a separate process or a step that always happens. Similarly, "re_submit_report" seems to be a loop back to the beginning of the process, but I'd like clarification on how it's triggered.

Please let me know and I'll be happy to help further.

After your feedback I will provide

Final Formal Declarative Constraints:

...

one constraint per line defined using the template provided.

User 1

⌚ You: create_account is an optional step, so you can decide if include it or not in the constraints. re_submit_report creates a loop going back to the point where a submit_report action happened, but this action "re_submit_report" can only happen after the request is not finished within 30 days and the client has received email cancellation notification.

LLM 2

💻 AI: Based on the process description and your feedback, I have identified the following activities:

- * receive_expense_report
- * create_account
- * review_report
- * automatic_approval
- * supervisor_approval
- * send_rejection_notice
- * send_approval_in_progress_email
- * send_cancellation_notice
- * reimbursement
- * re_submit_report

Here are my final declarative constraints:

Final Formal Declarative Constraints:

```
existence(review_report)  
response(review_report, automatic_approval)  
response(review_report, supervisor_approval)  
precedence(supervisor_approval, reimbursement)  
response(supervisor_approval, reimbursement)  
response(automatic_approval, reimbursement)
```

```
not-co-existence(send_rejection_notice, reimbursement)
response(send_rejection_notice, re_submit_report)
existence(send_cancellation_notice)
response(send_cancellation_notice, re_submit_report)
not-succession(reimbursement, send_cancellation_notice)
response(review_report, send_approval_in_progress_email)
precedence(send_cancellation_notice, re_submit_report)
```

Activities: receive_expense_report, create_account, review_report, automatic_approval, supervisor_approval, send_rejection_notice, send_approval_in_progress_email, send_cancellation_notice, reimbursement, re_submit_report

Note that I did not include "create_account" in the constraints as it seems to be an optional step. I also assumed that "receive_expense_report" is the starting point of the process, but I did not include it in the constraints as it seems to be an implicit start event. If you'd like to include it, please let me know.

Let me know if these constraints meet your requirements.

Process 43

AI: Based on the provided process description, I will attempt to translate it into declarative constraints.

First, let's identify the activities:

- * GO requests the measurements to the MSP: request_measurements
- * MSP checks the received request: check_request
- * MSP denies the request of the GO: deny_request
- * MSP performs the measurement: perform_measurement
- * MSP informs the GO about the failure of the reading: inform_failure
- * MSP transmits the measured values to the GO: transmit_values
- * GO processes the measured values: process_values
- * GO sends the changed values to the MSP: send_changed_values
- * GO transmit the readings to the SP: transmit_readings

Here are my temporary declarative constraints:

- * The MSP checks the request before denying it or performing the measurement: precedence(check_request, deny_request)
- * The MSP checks the request before transmitting the measured values or informing about failure: precedence(check_request, transmit_values), precedence(check_request, inform_failure)
- * The MSP either denies the request or performs the measurement: not-co-existence(deny_request, perform_measurement)
- * If the MSP performs the measurement, then it transmits the measured values: response(perform_measurement, transmit_values)
- * If the MSP informs about failure, then it does not transmit values: not-succession(inform_failure, transmit_values)
- * The GO processes the measured values after receiving them: response(transmit_values, process_values)
- * The GO sends changed values to the MSP after processing them: response(process_values, send_changed_values)

To better understand the process, I have a question:

Is it possible for the GO to send changed values to the MSP without processing the measured values first?

Assuming the answer is no, here are my final declarative constraints:

...

Final Formal Declarative Constraints:

```
precedence(check_request, deny_request)
precedence(check_request, perform_measurement)
precedence(check_request, transmit_values)
precedence(check_request, inform_failure)
not-co-existence(deny_request, perform_measurement)
response(perform_measurement, transmit_values)
not-succession(inform_failure, transmit_values)
response(transmit_values, process_values)
response(process_values, send_changed_values)
```

Activities: request_measurements, check_request, deny_request, perform_measurement, inform_failure, transmit_values, process_values, send_changed_values, transmit_readings

...

Approach 2

Process 15

AI: To translate the given process description into declarative constraints, let's first identify the activities mentioned:

- Finalize dates
- Update group calendars
- Create meeting folders
- Upload documents
- Advise Committee Members
- Hold meeting
- Produce minutes
- Conduct QA check
- Send minutes
- Update departmental records

The process description can be broken down into several steps, and we'll identify the relationships between activities to formulate declarative constraints.

Step 1: Identify the Start of the Process

The process starts with finalizing dates.

Step 2: Determine Sequential Activities

- Finalize dates leads to update group calendars and create meeting folders.
- Update group calendars and create meeting folders happen before upload documents.
- Upload documents happens before advise Committee Members.
- Advise Committee Members happens before hold meeting.
- Hold meeting happens before produce minutes.
- Produce minutes happens before conduct QA check.
- Conduct QA check happens before send minutes.
- Send minutes happens before update departmental records.

Step 3: Formulate Constraints

Based on the sequence and relationships:

- precedence(finalize_dates, update_group_calendars)
- precedence(finalize_dates, create_meeting_folders)
- response(finalize_dates, upload_documents)

- response(update_group_calendars, upload_documents)
- response(create_meeting_folders, upload_documents)
- response(upload_documents, advise_committee_members)
- response(advise_committee_members, hold_meeting)
- response(hold_meeting, produce_minutes)
- response(produce_minutes, conduct_qa_check)
- response(conduct_qa_check, send_minutes)
- response(send_minutes, update_departmental_records)
- succession(produce_minutes, conduct_qa_check)
- succession(conduct_qa_check, send_minutes)
- succession(send_minutes, update_departmental_records)

Step 4: Consider Other Constraints

- Init(finalize_dates) because the process starts with finalizing dates.

Step 5: Compile Activities and Constraints

Activities: finalize_dates, update_group_calendars, create_meeting_folders, upload_documents, advise_committee_members, hold_meeting, produce_minutes, conduct_qa_check, send_minutes, update_departmental_records

Step 6: Final Constraints

Given the complexity and the need for a concise output, let's focus on the critical path and direct successions.

Final Formal Declarative Constraints:

- Init(finalize_dates)
- precedence(finalize_dates, update_group_calendars)
- precedence(finalize_dates, create_meeting_folders)
- response(finalize_dates, upload_documents)
- response(upload_documents, advise_committee_members)
- response(advise_committee_members, hold_meeting)
- response(hold_meeting, produce_minutes)
- response(produce_minutes, conduct_qa_check)
- response(conduct_qa_check, send_minutes)
- response(send_minutes, update_departmental_records)
- succession(produce_minutes, conduct_qa_check)
- succession(conduct_qa_check, send_minutes)
- succession(send_minutes, update_departmental_records)

Activities: finalize_dates, update_group_calendars, create_meeting_folders, upload_documents, advise_committee_members, hold_meeting, produce_minutes, conduct_qa_check, send_minutes, update_departmental_records

Process 35

AI: To translate the given process description into declarative constraints, let's first identify the activities mentioned:

- Receiving the Expense Report
- Creating a new account
- Reviewing the report
- Automatic approval
- Approval of the supervisor
- Rejection
- Sending a rejection notice by email
- Reimbursement to the employee's direct deposit bank account
- Sending an approval in progress email
- Stopping the process
- Sending an email cancellation notice
- Re-submitting the expense report

Let's denote these activities with simpler terms for clarity:

- expense_report_received
- create_account
- review_report
- automatic_approval
- supervisor_approval
- rejection
- send_rejection_notice
- reimbursement
- send_approval_in_progress_email
- stop_process
- send_cancellation_notice
- resubmit_expense_report

Based on the process description, here are the declarative constraints:

1. The process starts with receiving the expense report.

- Init(expense_report_received)

2. If the expense report is received, then a new account must be created if the employee does not already have one.

- response(expense_report_received, create_account)

3. The report is then reviewed for automatic approval.

- succession(expense_report_received, review_report)

4. Amounts under \$200 are automatically approved.

- automatic_approval

5. Amounts equal to or over \$200 require approval of the supervisor.

- response(review_report, supervisor_approval)

6. In case of rejection, the employee must receive a rejection notice by email.

- response(rejection, send_rejection_notice)

7. If the request is not completed in 7 days, then the employee must receive an approval in progress email.

- not-succession(review_report, reimbursement) and response(review_report, send_approval_in_progress_email)

8. If the request is not finished within 30 days, then the process is stopped and the employee receives an email cancellation notice and must re-submit the expense report.

- not-succession(review_report, reimbursement) and response(review_report, stop_process) and response(stop_process, send_cancellation_notice) and response(send_cancellation_notice, resubmit_expense_report)

However, these can be simplified and directly related to the given templates:

Temporary Declarative Constraints:

- Init(expense_report_received)
- response(expense_report_received, create_account)
- succession(expense_report_received, review_report)
- response(review_report, automatic_approval)
- response(review_report, supervisor_approval)
- response(rejection, send_rejection_notice)
- not-response(rejection, reimbursement)
- response(review_report, send_approval_in_progress_email)
- not-chain-response(review_report, reimbursement)

- response(review_report, stop_process)
- response(stop_process, send_cancellation_notice)
- response(send_cancellation_notice, resubmit_expense_report)

****Final Formal Declarative Constraints:****

- Init(expense_report_received)
- response(expense_report_received, create_account)
- succession(expense_report_received, review_report)
- at-most(supervisor_approval)
- response(rejection, send_rejection_notice)
- not-response(rejection, reimbursement)
- not-succession(review_report, reimbursement)
- response(review_report, send_approval_in_progress_email)
- not-chain-response(review_report, reimbursement)
- response(review_report, stop_process)
- response(stop_process, send_cancellation_notice)
- response(send_cancellation_notice, resubmit_expense_report)
- succession(rejection, send_rejection_notice)
- existence(send_rejection_notice)
- existence(reimbursement)
- existence(send_approval_in_progress_email)
- existence(send_cancellation_notice)

****Activities:**** expense_report_received, create_account, review_report, automatic_approval, supervisor_approval, rejection, send_rejection_notice, reimbursement, send_approval_in_progress_email, stop_process, send_cancellation_notice, resubmit_expense_report

Process 43

AI: To translate the given process description into declarative constraints, let's first identify the activities involved:

- GO requests the measurements to the MSP: request_measurement
- MSP checks the received request: check_request
- MSP denies the request of the GO: deny_request
- MSP performs the measurement: perform_measurement
- MSP informs the GO about the failure of the reading: inform_failure
- MSP transmits the measured values to the GO: transmit_measured_values
- GO processes the measured values: process_measured_values
- GO sends the changed values to the MSP: send_changed_values
- GO transmits the readings to the SP: transmit_readings

Activities: request_measurement, check_request, deny_request, perform_measurement, inform_failure, transmit_measured_values, process_measured_values, send_changed_values, transmit_readings

Based on the process description, we can derive the following sequence and constraints:

1. The process starts with `request_measurement`.

- Init(request_measurement)

2. `check_request` must occur after `request_measurement` and before either `deny_request` or `perform_measurement`.

- precedence(check_request, deny_request)
- precedence(check_request, perform_measurement)

3. If `deny_request` occurs, it must be after `check_request`.

- response(check_request, deny_request)

4. If `perform_measurement` occurs, it must be after `check_request`.

- response(check_request, perform_measurement)

5. After `perform_measurement`, either `inform_failure` or `transmit_measured_values` must occur.

- exclusive-choice(inform_failure, transmit_measured_values)

6. If `transmit_measured_values` occurs, then `process_measured_values` must occur after it.

- response(transmit_measured_values, process_measured_values)

7. `send_changed_values` must occur after `process_measured_values`.

- response(process_measured_values, send_changed_values)

8. Finally, `transmit_readings` must occur after `send_changed_values`.

- response(send_changed_values, transmit_readings)

However, upon closer inspection and to adhere strictly to the provided templates and to ensure accuracy, let's refine and directly list the constraints based on the activities and their logical sequence:

Final Formal Declarative Constraints:

- Init(request_measurement)
- precedence(check_request, deny_request)
- precedence(check_request, perform_measurement)
- response(check_request, deny_request)
- response(check_request, perform_measurement)
- exclusive-choice(inform_failure, transmit_measured_values)
- response(transmit_measured_values, process_measured_values)
- response(process_measured_values, send_changed_values)
- response(send_changed_values, transmit_readings)
- succession(request_measurement, check_request)
- succession(perform_measurement, inform_failure)
- succession(perform_measurement, transmit_measured_values)
- succession(transmit_measured_values, process_measured_values)
- succession(process_measured_values, send_changed_values)
- succession(send_changed_values, transmit_readings)

Activities: request_measurement, check_request, deny_request, perform_measurement, inform_failure, transmit_measured_values, process_measured_values, send_changed_values, transmit_readings

Approach 3

Process 15

AI: To translate the given process description into declarative constraints, let's first identify the activities mentioned:

- Finalize dates
- Update group calendars
- Create meeting folders
- Upload documents
- Advise Committee Members
- Hold meeting
- Produce minutes
- Conduct QA check
- Send minutes
- Update departmental records

The process description can be broken down into several steps, and from these, we can infer the following constraints:

1. ****Init(finalize_dates)**:** The process starts with finalizing dates.
2. ****Succession(finalize_dates, update_group_calendars)**:** Finalizing dates is followed by updating group calendars.
3. ****Succession(update_group_calendars, create_meeting_folders)**:** Updating group calendars is followed by creating meeting folders.
4. ****Succession(create_meeting_folders, upload_documents)**:** Creating meeting folders is followed by uploading documents.
5. ****Response(advise_committee_members, hold_meeting)**:** Advising Committee Members occurs before the meeting.
6. ****Succession(hold_meeting, produce_minutes)**:** The meeting is followed by producing minutes.
7. ****Precedence(produce_minutes, conduct_qa_check)**:** Conducting a QA check occurs after producing minutes.
8. ****Response(produce_minutes, conduct_qa_check)**:** Producing minutes implies conducting a QA check.
9. ****Succession(conduct_qa_check, send_minutes)**:** The QA check is followed by sending minutes.
10. ****Succession(send_minutes, update_departmental_records)**:** Sending minutes is followed by updating departmental records.
11. ****Existence(finalize_dates)**:** Finalizing dates must occur at least once.
12. ****Existence(update_group_calendars)**:** Updating group calendars must occur at least once.
13. ****Existence(create_meeting_folders)**:** Creating meeting folders must occur at least once.
14. ****Existence(upload_documents)**:** Uploading documents must occur at least once.

15. **Existence(advice_committee_members)**: Advising Committee Members must occur at least once.
16. **Existence(hold_meeting)**: Holding the meeting must occur at least once.
17. **Existence(produce_minutes)**: Producing minutes must occur at least once.
18. **Existence(conduct_qa_check)**: Conducting a QA check must occur at least once.
19. **Existence(send_minutes)**: Sending minutes must occur at least once.
20. **Existence(update_departmental_records)**: Updating departmental records must occur at least once.

However, upon analyzing the implications provided:

- The strongest constraints that imply others should be selected.

Given the process and constraints:

****Activities:** finalize_dates, update_group_calendars, create_meeting_folders, upload_documents, advise_committee_members, hold_meeting, produce_minutes, conduct_qa_check, send_minutes, update_departmental_records**

****Final Formal Declarative Constraints:****

1. Init(finalize_dates)
2. Succession(finalize_dates, update_group_calendars)
3. Succession(update_group_calendars, create_meeting_folders)
4. Succession(create_meeting_folders, upload_documents)
5. Response(advice_committee_members, hold_meeting)
6. Succession(hold_meeting, produce_minutes)
7. Precedence(produce_minutes, conduct_qa_check)
8. Succession(conduct_qa_check, send_minutes)
9. Succession(send_minutes, update_departmental_records)
10. Existence(finalize_dates)
11. Existence(hold_meeting)
12. Existence(produce_minutes)
13. Existence(conduct_qa_check)

These constraints capture the essential flow and requirements of the process described, focusing on the mandatory sequences and existences without redundancy.

Note that some of the existence constraints for every single activity could potentially be implied or are stronger than needed given the succession and response constraints provided. However, given the direct description, ensuring each step happens is crucial. Adjustments might be necessary based on a deeper analysis of process implications and dependencies.

The selection here directly translates from the process description into formal declarative constraints, ensuring that the process flows as intended while adhering to the specified format and implications.

Process 35

AI: To translate the given process description into declarative constraints, let's first identify the activities involved:

- Receiving Expense Report
- Creating new account
- Reviewing report
- Automatic approval
- Approval by supervisor
- Rejection
- Sending rejection notice
- Reimbursement to direct deposit
- Sending approval in progress email
- Sending cancellation notice

Now, let's analyze the process description to extract the constraints:

1. **After the Expense Report is received, a new account must be created if the employee does not already have one.**

- This implies that "Creating new account" must occur after "Receiving Expense Report" if certain conditions are met, suggesting a **precedence** relationship but with a condition. However, directly translating this into a given template, we focus on the existence and succession:

2. **The report is then reviewed for automatic approval.**

- **Reviewing report** must occur.

3. **Amounts under \$200 are automatically approved, whereas amounts equal to or over \$200 require approval of the supervisor.**

- This suggests a **choice** between **Automatic approval** and **Approval by supervisor**.

4. **In case of rejection, the employee must receive a rejection notice by email.**

- If **Rejection** occurs, then **Sending rejection notice** must occur.

5. **Otherwise, the reimbursement goes to the employees direct deposit bank account.**

- Suggests that if not **Rejection**, then **Reimbursement to direct deposit**.

6. **If the request is not completed in 7 days, then the employee must receive an approval in progress email.**

- Implies a **response** relationship: If the request is not completed, **Sending approval in progress email** must occur.

7. **If the request is not finished within 30 days, then the process is stopped and the employee receives an email cancellation notice and must re-submit the expense report.**

- Implies **not-response** and **not-succession** constraints for the process stopping.

Given these points, let's draft some initial constraints:

- **existence(Reviewing report)**
- **choice(Automatic approval, Approval by supervisor)**
- **response(Rejection, Sending rejection notice)**
- **not-response(Rejection, Reimbursement to direct deposit)**
- **response(not completed within 7 days, Sending approval in progress email)**
- **not-succession(not finished within 30 days, Reimbursement to direct deposit)**
- **not-response(not finished within 30 days, Cancellation notice)**
- **precedence(Receiving Expense Report, Creating new account)**

However, let's refine and accurately map these to provided templates and ensure we capture the essence:

Activities: Receiving Expense Report, Creating new account, Reviewing report, Automatic approval, Approval by supervisor, Rejection, Sending rejection notice, Reimbursement to direct deposit, Sending approval in progress email, Sending cancellation notice

Temporary Declarative Constraints:

- existence(Reviewing report)
- choice(Automatic approval, Approval by supervisor)
- response(Rejection, Sending rejection notice)
- not-response(Rejection, Reimbursement to direct deposit)
- response(Absence of completion within 7 days, Sending approval in progress email)
- not-response(Absence of completion within 30 days, Reimbursement to direct deposit)
- precedence(Receiving Expense Report, Creating new account)

Final Formal Declarative Constraints:

- existence(Reviewing report)
- choice(Automatic approval, Approval by supervisor)
- response(Rejection, Sending rejection notice)
- not-response(Rejection, Reimbursement to direct deposit)
- response(not completed within 7 days, Sending approval in progress email)
- not-response(not finished within 30 days, Reimbursement to direct deposit)

- precedence(Receiving Expense Report, Creating new account)
- not-exclusive-choice(Sending approval in progress email, Sending cancellation notice)

Given the complexity and ensuring clarity, let's finalize:

Activities: Receiving Expense Report, Creating new account, Reviewing report, Automatic approval, Approval by supervisor, Rejection, Sending rejection notice, Reimbursement to direct deposit, Sending approval in progress email, Sending cancellation notice

Final Formal Declarative Constraints:

- existence(Reviewing report)
- choice(Automatic approval, Approval by supervisor)
- response(Rejection, Sending rejection notice)
- precedence(Receiving Expense Report, Creating new account)
- not-response(Rejection, Reimbursement to direct deposit)
- response(not completed within 7 days, Sending approval in progress email)
- not-response(not finished within 30 days, Reimbursement to direct deposit)

Process 43

AI: To translate the given process description into declarative constraints, let's first identify the activities involved:

- GO requests the measurements to the MSP: Request_Measurements
- MSP checks the received request: Check_Request
- MSP denies the request of the GO: Deny_Request
- MSP performs the measurement: Perform_Measurement
- MSP informs the GO about the failure of the reading: Inform_Failure
- MSP transmits the measured values to the GO: Transmit_Measured_Values
- GO processes the measured values: Process_Measured_Values
- GO sends the changed values to the MSP: Send_Changed_Values
- GO transmits the readings to the SP: Transmit_Readings

Activities: Request_Measurements, Check_Request, Deny_Request, Perform_Measurement, Inform_Failure, Transmit_Measured_Values, Process_Measured_Values, Send_Changed_Values, Transmit_Readings

Based on the process description, we can derive the following steps and constraints:

1. **Request_Measurements** by GO is the starting point.
 - Init(Request_Measurements)
2. **Check_Request** must happen after **Request_Measurements**.
 - Precedence(Request_Measurements, Check_Request)
3. After **Check_Request**, either **Deny_Request** or **Perform_Measurement** happens.
 - Exclusive-Choice(Deny_Request, Perform_Measurement)
4. If **Deny_Request**, then **Inform_Failure** must happen next.
 - Response(Deny_Request, Inform_Failure)
5. If **Perform_Measurement**, then **Transmit_Measured_Values** must happen next.
 - Response(Perform_Measurement, Transmit_Measured_Values)
6. **Process_Measured_Values** happens after **Transmit_Measured_Values**.
 - Precedence(Transmit_Measured_Values, Process_Measured_Values)
7. **Send_Changed_Values** happens after **Process_Measured_Values**.
 - Precedence(Process_Measured_Values, Send_Changed_Values)
8. **Transmit_Readings** happens after **Send_Changed_Values**.
 - Precedence(Send_Changed_Values, Transmit_Readings)
9. **Inform_Failure** does not lead to **Process_Measured_Values** directly; it seems to end the process or lead to another path not described, implying:
 - Not-Succession(Inform_Failure, Process_Measured_Values)

However, upon closer inspection and to maintain simplicity and adherence to provided templates and implications:

Temporary Declarative Constraints:

- Init(Request_Measurements)
- Precedence(Request_Measurements, Check_Request)
- Exclusive-Choice(Deny_Request, Perform_Measurement)
- Response(Deny_Request, Inform_Failure)
- Response(Perform_Measurement, Transmit_Measured_Values)
- Precedence(Transmit_Measured_Values, Process_Measured_Values)
- Precedence(Process_Measured_Values, Send_Changed_Values)
- Precedence(Send_Changed_Values, Transmit_Readings)
- Not-Succession(Inform_Failure, Process_Measured_Values)

After applying implications:

- The stronger constraints that subsume weaker ones should be kept.

Final Formal Declarative Constraints:

- Init(Request_Measurements)
- Precedence(Request_Measurements, Check_Request)
- Exclusive-Choice(Deny_Request, Perform_Measurement)
- Response(Deny_Request, Inform_Failure)
- Response(Perform_Measurement, Transmit_Measured_Values)
- Precedence(Transmit_Measured_Values, Process_Measured_Values)
- Precedence(Process_Measured_Values, Send_Changed_Values)
- Precedence(Send_Changed_Values, Transmit_Readings)
- Not-Succession(Inform_Failure, Process_Measured_Values)

Activities: Request_Measurements, Check_Request, Deny_Request, Perform_Measurement, Inform_Failure, Transmit_Measured_Values, Process_Measured_Values, Send_Changed_Values, Transmit_Readings