Faculteit Bedrijfseconomische Wetenschappen

master handelsingenieur in de beleidsinformatica

Masterthesis

Exploring the Impact of DECLARE Constraint Modifications on Data Programming in a Business Process Context

Torsten Daniels

Scriptie ingediend tot het behalen van de graad van master handelsingenieur in de beleidsinformatica

PROMOTOR :
Prof. dr. Mieke JANS

BEGELEIDER :
Mevrouw Manal LAGHMOUCH

2022
2023

Faculteit Bedrijfseconomische Wetenschappen

master handelsingenieur in de beleidsinformatica

Masterthesis

Exploring the Impact of DECLARE Constraint Modifications on Data Programming in a Business Process Context

Torsten Daniels

Scriptie ingediend tot het behalen van de graad van master handelsingenieur in de beleidsinformatica

PROMOTOR :

Prof. dr. Mieke JANS

BEGELEIDER :

Mevrouw Manal LAGHMOUCH

Exploring the Impact of DECLARE Constraint Modifications on Data Programming in a Business Process Context

Torsten Daniels[1], Prof. dr. Mieke Jans[1], and Manal Laghmouch[1]

[1]Department of business informatics, Hasselt University,
Agoralaan Building D, Diepenbeek, 3590, Belgium

Auditors analyse process deviations in companies to search for anomalous behaviour. Machine learning can help an auditor by efficiently testing the entire population of transactions, but labelled training data is required. However, recent research successfully used a machine learning paradigm that does not need this training data. Instead, rules describing the allowed behaviour were used. A reasonable result was achieved with only a few medium- or high-quality rules. However, methodological approaches have yet to be known to achieve these rules. We conducted an explorative experiment to assess the impact of strengthening or weakening rules on the machine learning model's performance. This examination was to find correlations that can be considered when future research tries to develop a methodology to obtain qualitative rules. A rule hierarchy allows us to strengthen or weaken rules individually. This was done in five different scenarios. In our study, we did not find a correlation between strengthening or weakening rules and the accuracy or precision of a machine learning model, nor did these modifications result in a predictable change of a modified rule's accuracy. However, our study did find a correlation between strengthening constraints and increased recall of the machine learning model. We also confirmed that only a small set of medium- or high-quality constraints are needed to achieve a machine-learning model with reasonable performance. However, since the experiment was conducted only on the procure-to-pay process, we suggest reviewing the findings with other business processes in future research. These can findings help develop a methodological approach to creating machine learning models for audit purposes in a business process context.

Keywords - *Data programming, Labeling, Process deviations, Weak Supervision, DECLARE constraints, hierarchy of DECLARE constraints*

Acknowledgements: I want to express my sincere gratitude to my supervisors, Prof. Dr. Jans and Manal Laghmouch, for their unwavering support and guidance throughout my thesis. Furthermore, I would like to thank my partner, Eline Maesen, for her encouragement and motivation to keep me going.

I Introduction

Mature organisations have business processes in place that allow them to work standardised. These processes describe the sequence and content of employees' tasks. It allows organisations to implement automatisation. Business processes make managing product or service quality easier since we can identify and implement improvements within a process (Cugola, 1998). However, these processes also restrict companies in their actions. Therefore, these processes act like guidelines to allow some flexibility. Actions that do not follow the business process strictly are called *process deviations*. Although some process deviations are acceptable, others are mistakes during the execution or indicate fraud (Swinnen et al., 2011).

Auditors are interested in these process deviations. Since the digital transformation within companies, processes are increasingly digital-based (Nonnenmacher & Marx Gómez, 2021). This increase implies that companies generate more data than before. To perform an audit, auditors must sample the data to conclude the entire population. The International Standard on Auditing 530 (ISA 530) states that the sample set of items must have a reasonable basis for this. However, new technologies could assist an auditor in testing the entire population of transactions.

Process mining, for example, offers the ability to research company event logs. They can extract the business process from these event logs and compare it to a normative process model. The *normative process model* is the business process model a company wants or needs to follow. It is even possible to compare the events against different business rules that reflect the business process in a declarative way. This comparison is called conformance checking. *Conformance checking* investigates if all events comply with the business rules. If events are non-compliant, we can label these as variants of the business process. However, a thorough analysis of the variants is still needed to determine whether the process deviation is an exception or anomalous behaviour (Jans & Laghmouch, 2023).

Machine learning can automate the analysis of variants while auditing the entire population instead of a sample. In addition, machine learning can speed up and increase the quality of audits (Dickey et al., 2019). We can train machine learning models to distinguish between exceptional or anomalous behaviour. However, machine learning algorithms need training data. Obtaining and labelling this data to train a machine learning model would still be time-consuming and expensive (Ratner et al., 2016). Thus, to help auditors using machine learning, we must find a way to generate labelled, artificial data to train machine learning algorithms for auditing purposes.

Data programming is a machine learning paradigm that enables people to generate training data for machine learning models using noisy rules. These rules are statements about the data set but can be partially correct. In data programming, knowing the accuracy of such a rule is optional as long as a rule makes sense to the user (Ratner et al., 2016). For example, a rule that states it is anomalous behaviour when a purchase order is not signed off can be correct. However, it is only valid if the process has already performed further activities. The rule is wrong if the process is awaiting a signature. The strength of this paradigm is that the data programming model relies on weak supervision. *Weak supervision* means the model does not need a labelled dataset as a training set and, thus, does not need access to ground truth. A proof of concept using data programming demonstrates the potential with only a limited amount of noisy rules (Laghmouch, Jans, & Depaire, 2020). In this study, the researchers used random rules in different iterations to test the usability. However, a formal method to create such rules has yet to exist.

In this paper, we will experiment with rule modification according to the rule hierarchy of Schunselaar et al. (2012). This hierarchy shows how rules can be adapted to reflect stricter behaviour (strengthening) or more lenient behaviour (weakening). We will review the impact of such modifications on the accuracy of the rule and the data programming model. We will identify research gaps that need further investigation to better understand data programming in the context of business processes. Insights acquired from investigating these research gaps might lead researchers to a methodological approach in determining sets of constraints that result in highly accurate data programming models.

This experiment suggests that strengthening or weakening rules does not significantly impact the data programming model, nor does it influence the accuracy of rules in a controlled manner. This implies that the rule hierarchy cannot be used to create or modify rules to obtain a set of constraints for an accurate data programming model. However, there is a correlation between strengthened constraints and how well our data programming model identifies all positive cases (the recall). Furthermore, we can confirm the study of Laghmouch et al. (2020). We also achieved reasonable performance of the data programming model with enough medium- or high-quality rules. With future research, we might extrapolate these findings to other business processes. By doing so, we can find out what identifies good heuristics and, ideally, help find a methodological approach to obtaining qualitative rules to train data programming models to help auditors in the field.

The paper is structured as follows. Section II will briefly discuss data programming, process deviations, rule expressions, and the hierarchy of rules. We will describe the method used to modify rules and analyse the impact in Section III. The results are then shown and discussed in Section IV and Section V. The paper will propose future work in Section VI, and conclude in Section VII.

II Related work

This section provides work related to data labelling, process deviation analysis, and rules in the context of business processes. The first part introduces data programming. It is a machine-learning paradigm that aims to train models using heuristics without giving the model access to labels in a training set. Next, we discuss process deviations and their different types. It helps us understand what deviations we want to identify during an audit. The following section will describe different representations of process models to use in our data programming model. Finally, we will discuss the difference between constraint hierarchies in a declarative and procedural context.

A Data programming

In machine learning, we aim to train models to give us valuable insights into the abundance of available data. To train such models, we rely on training sets of data that are all labelled. This labelling can be resourceful, expensive, and time-consuming (Cunningham et al., 2008)(Ratner et al., 2019).

Data programming is a programming paradigm that aims to solve the problem of creating training data for machine learning without manually labelling each observation in a dataset (Ratner et al., 2016). Instead, observations are labelled based on heuristics. These heuristics predict the label of a data point based on its features. For example, a heuristic based on the presence of a signature on a purchase order can be used to classify processes into correct or incorrect behaviour. However, heuristics can be noisy, meaning they can misclassify some observations. In the previous example, the heuristic would only be correct if the process continued without the signature. If the process is awaiting the signature, then the execution of the process is still allowed. Combining multiple noisy heuristics can increase the accuracy of predictions (Ratner et al., 2017). Heuristics are assigned an accuracy score, and a correlation structure is created between them to assign a final label. Data programming uses these properties to train a machine learning model and make it noise-aware (Ratner et al., 2016). By minimising the need for hand-labelled data, data programming reduces the costs of creating a training dataset (Ratner et al., 2017).

Snorkel is a Python library that uses data programming to generate a training dataset. It allows the generation of an artificially labelled dataset without manually labelling each case in the dataset. When we train machine learning models using noisy labelled datasets, we call this weakly supervised machine learning or machine learning under weak supervision. Hence, the output of data programming can be used for weak supervision.

Laghmouch et al. (2020) applied data programming in an artificial auditing setting. The authors show that data programming has the potential to classify process deviations. Only a few medium- or high-quality heuristics were needed to create a classification model with acceptable accuracy. The quality of heuristics depends on the accuracy of the heuristic to classify data individually. Although Laghmouch et al. (2020) used heuristics representing an auditor's knowledge, no step-by-step guide was used to create qualitative rules. Finding a method that conveniently creates qualitative rules will allow data programming in auditing practice by domain experts (Laghmouch et al., 2020).

This paper explores what influences the accuracy of heuristics and, thus, their quality. In addition, we will explore how this reflects on the performance of data programming. Future research can then consider these results when searching for a methodological approach to create qualitative rules.

B Process deviation

A process deviation is a difference between an as-is process and the normative process model. The *as-is process* is the actual flow of activities a company performs when executing a process. A company records this as a log. The *normative process model* is the desired procedure a company wants to follow (Zuhaira et al., 2017). For example, the normative process model can be a best practice or a standard.

While some deviations allow a company to be flexible, others indicate that a process containing errors, flaws, or fraud is present within a company. When these deviations also potentially impact the financial statements, we call them *anomalies*. These anomalies are what interest an auditor. An auditor wants to explain these anomalies to reassure that the business results reflect reality. We can split the remaining deviations among explicit exceptions and implicit exceptions (Swinnen et al., 2011). *Explicit* *exceptions* are those generally accepted by the company and where guides are present to handle them. *Implicit exceptions* do not have these guides; however, they are allowed occasionally. This categorisation of deviations implies that not all process deviations should be considered malpractice. However, detecting whether a deviation is an anomaly or an implicit exception is hard. A thorough understanding of the process is needed to make this decision.

This study will focus on deviations in a business process context. We aim to explore how we can modify heuristics to classify the process deviations as accurately as possible between anomalous and allowed (exceptional) behaviour.

C Representations of Process Models in a Business Process Context

A step-by-step procedure can describe business processes. This is called an imperative approach. This approach specifies every step or activity that should be followed but leaves no space for deviations since every decision is made when designing the process model (Pichler et al., 2011). An example is Petri Net. The following example applies the imperative approach on the processing of a payroll. Every action an automation or human performs is encapsulated in a step-by-step procedure. This results in everyone being paid after all steps are followed. However, by only providing one step-by-step procedure, everyone will be paid the same amount.

Flexible systems try to overcome this by allowing users to make decisions instead of the process model. Systems like FLOWer allow flexible behaviour but still use the imperative approach (Pesic, Schonenberg, & van der Aalst, 2007). This means the user will still need to perform the activities prescribed in the process model. However, implementing decision points in our previous example allows us to pay employees according to their function. Certain functions might get a car, while other functions receive a laptop. However, the possible decisions are still limited to the ones described in the model. Allowing decision points in the process model creates the opportunity to differentiate executions from one another, but the activities' flow is still encapsulated for each decision.

Where an imperative approach provides a step-by-step procedure, a declarative approach provides a set of constraints that should always be followed. The declarative model can be considered a playground where the constraints are the boundaries of it. If a boundary is crossed, at least one constraint is violated. Returning to the example, we can describe boundaries to what certain functions can get paid. This way, everyone can be paid according to their unique contract if the boundaries are not crossed. Hence, the execution of the process can be different at all times and be adapted to different needs. DECLARE represents process models and uses the declarative approach by defining *DECLARE constraints* (Pesic et al., 2007).

DECLARE allows companies to do their finance tailored to their needs as long as they adhere to the rules set up by, i.e., the government. Auditors will then audit to see if the company follows the rules while not enforcing a specific way of working. The company remains flexible, and the auditor can perform his work. Therefore, DECLARE constraints allow us to describe the allowed behaviour and use them as heuristics for our data programming model.

D Hierarchy of rules

Introducing a rule hierarchy helps us better interpret how to handle rules. More specifically, it can provide information on prioritising or adapting rules to suit our needs. In the context of DECLARE constraints, we can hierarchise rules differently.

The first hierarchy is the procedural constraint hierarchy, which enables us to prioritise rules over one another. Constraints receive a particular priority, which indicates whether a constraint should hold above all others or can be ignored if a more critical constraint still needs to be satisfied (Borning et al., 1989). Constraints with a lower priority will be more lenient and thus more easily violated than those with a higher priority.

Schunselaar et al. (2012) describe the hierarchy of *DECLARE* constraints in their research. The hierarchy defines that constraints can be weakened or strengthened (Schunselaar et al., 2012). Instead of making a hierarchy between different constraints, we have a hierarchy among the different constraint types. This hierarchy shows that some constraint types are stricter or looser than others, implying we can strengthen or weaken DECLARE constraints individually and thus change the allowed behaviour.

In this paper, we are interested in how allowing more or less behaviour impacts the performance of a data programming model. This will be done by modifying rules separately. The procedural constraint hierarchy does not allow this since prioritisation modifies all rules. In addition, data programming already implements its prioritisation technique by assigning weights to the different rules based on their performance. The DECLARE constraint hierarchy does allow us to modify constraints individually without influencing others. Hence, we will modify constraints using the DECLARE constraint hierarchy of Schunselaar et al. (2012).

III Method

We will conduct an explorative experiment to investigate whether there is a relationship between the hierarchy of DECLARE constraints and the accuracy of data programming. The aim is to provide insights and starting points for future research for data programming in a business process context.

We will train a data programming model using Snorkel and conduct our research in an artificial audit context. More specifically, this experiment will compromise data from a procure-to-pay process. However, the data does not originate from authentic company data. Our data is generated using an *auditor process model*. This model is a subset of the normative process model and represents the constraints that will correctly classify cases of a process model as anomalous or allowed (exceptional) behaviour. However, as they are used to generate the data, a different process model is used as the baseline for our experiment.

The baseline for our data programming model is a process model that exists out of a set of constraints that enforces the flow of activities just like the normative process model does. However, the relationship between activities is random and, thus, does not reflect the same type of relationship as the normative process model or the auditor process model. With this baseline model, we will train the data programming model to classify cases as anomalous or allowed. The constraints will also not allow the data programming model to abstain. This means that the model will always label a case as either anomalous or allowed. As best practices recommend, we will train on only a third of the cases in the dataset (Alpaydin, 2010). After training, we will calculate three different metrics of the data programming model on a test set. This test set exists out of the remaining cases from the original dataset and thus contains cases the algorithm has not seen before. The metrics are accuracy, recall, and precision.

Constraints can be strengthened or weakened. When we strengthen constraints, we allow less behaviour, while weakening them allows more behaviour. We will show the impact on the performance of the data programming model using five different scenarios:

- Scenario 0: This will be the baseline for our experiment. We will measure the accuracy of each constraint individually and the performance of the entire data programming model. All other scenarios will be compared to this scenario to see the impact of the constraint modifications we will perform in each of the following scenarios.

- Scenario 1: Using the constraint hierarchy of Schunselaar et al. (2012), we will strengthen each baseline constraint individually by one level. One level indicates that no DECLARE constraint exists between the strengthened and baseline constraint. The performance of the data programming model and the constraint will be compared to scenario 0.

- Scenario 2: The same will be done in this scenario. However, the baseline constraints will now be weakened individually. Also, for this scenario, the performance of both the constraint and data programming model will be compared to scenario 0.
- Scenario 3: We will check if the performance of the data programming model gradually improves when the baseline constraints are strengthened or weakened level-by-level until they become the constraints of the auditor process model. This will be done for each constraint individually while keeping the other constraints constant. The performance of the constraint and the data programming model will be measured throughout every modification and compared.
- Scenario 4: The last scenario shows the capabilities of data programming when using the constraints from the auditor process model.

By comparing the performance of the data programming model across different scenarios, we will look for a correlation between constraint modifications and the performance of the data programming model. These findings will be of use as starting points for future research.

A Procure-to-pay Process

This experiment will be applied to a procure-to-pay process. The auditor process model is a subset of the normative process model (Figure 1) and reflects the flow of important activities to the auditor. We adopt the auditor process model from the work of Laghmouch et al. (2020) (Figure 2). It is assumed that this model consists of all constraints and that when all constraints are satisfied, an auditor will label the process execution as allowed (exceptional) behaviour. An auditor would label it anomalous if one of the constraints is not satisfied. However, it will not be used to train our model. We will use a third process model to which we will refer as *the experiment process model* (Figure 3). Although the activities' flow is roughly the same as the normative process model, the relationship type between activities differs from the normative process model. These types are randomly chosen while keeping the model meaningful. Therefore, the experiment process model mimics what an auditor might use as a starting point for his process model. The constraints of this model will be modified to explore its impact on the performance of the data programming model. Therefore, it serves as the baseline for this experiment. It will be modified in accordance with the four scenarios described above to train the data programming model.

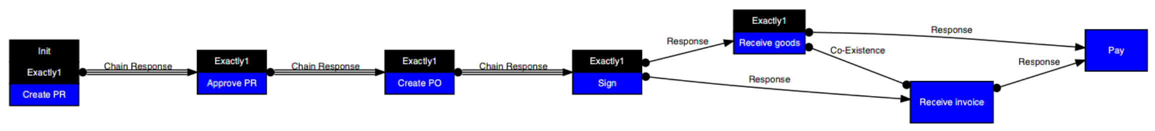


Figure 1: The normative process model

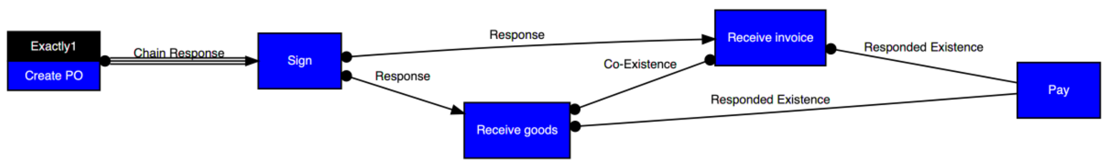


Figure 2: The auditor process model

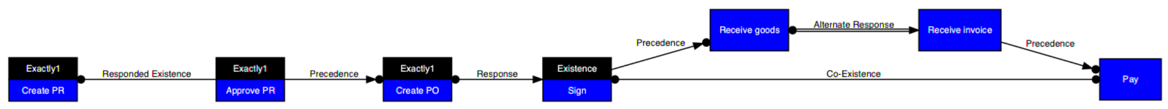


Figure 3: The experiment process model

B Data

An artificial dataset that reflects a declarative process model is generated using the RuM[1] application. *RuM* is an online tool that can create traces of events complying (or not complying) with a specific process model. We used the auditor process model for this experiment to generate 3000 cases. These cases are all process deviations and mimic the outcome of performing a conformance check. Based on the auditor process model, these deviations are split into two categories, which we will try to identify with data programming. Half of the cases are allowed (exceptional) behaviour, while the other half are anomalous. Every case consists of events. These events indicate which activity was executed, which case they belong to, when the activity was done, and the label to what class of behaviour the case belongs. The generated data was then split among two datasets. The first data set is the training set. This set consists of 2000 cases with an even distribution of behaviour. In order to remove the access to ground truth for the data programming model, we removed the class label. We did the same for the test set, taking the remaining 1000 cases from the generated dataset. However, this time we separated the class label instead

[1]https://rulemining.org

of removing it. This separation still denies the data programming model access to ground truth, but by keeping the labels, we can calculate the performance of the data programming model.

C Independant variables

Every constraint in the experiment process model is an independent variable for this experiment. The independent variables are the constraints we will modify during our experiment. We will strengthen or weaken them using the hierarchy of DECLARE constraints from Schunselaar et al. (2012).

Our experiment's baseline is the process model described in Figure 3. Table 1 shows the relationship between the activities. This also forms the base value of our independent variables. These independent variables will be strengthened or weakened according to the abovementioned scenarios.

Activity A	**Activity B**	**Baseline value - Relationship(A [, B])**
Create PR	Approve PR	Responded Existence(A, B)
Approve PR	Create PO	Precedence(A, B)
Create PO	Sign	Response(A, B)
Sign	receive goods	Precedence(A, B)
Receive goods	Receive invoice	Alternate Response(A, B)
Receive invoice	Pay	Precedence(A, B)
Receive goods	Receive invoice	Co-existence(A, B)
Create PR	/	Exactly(A, 1)
Approve PR	/	Exactly(A, 1)
Create PO	/	Exactly(A, 1)
Sign	/	Existence(A, 1)

Table 1: The independant variables

D Dependant variables (metrics)

We will use the metrics precision, recall, and accuracy to measure the performance of the data programming model. These are a means to indicate how well the data programming model performs (Vakili et al., 2020).

Precision tells us how many of the positive predictions made by the model are true positives. It is the ratio of the true positives to the total number of positive predictions (Hossin & M.N, 2015). For this experiment, a high precision score means that when the model marks a case as anomalous, it is highly confident it is anomalous.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

Recall measures how well our model can identify all anomalous cases. It is the ratio of the true positives to the total number of positive cases (Hossin & M.N, 2015). It is important for an auditor that this number is as high as possible because it allows him to trust the model to identify all non-compliant cases.

$$Recall = \frac{TP}{TP+FN} \tag{2}$$

Accuracy indicates how well the model performs overall. It is the ratio of correct predictions to the total number of predictions (Hossin & M.N, 2015). This metric calculates the performance of a data programming model in an easy-to-understand manner.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{3}$$

For all formulas, the TP stands for *true positives*, FP stands for *false positives*, TN stands for *true negatives*, and FN stands for *false negatives*.

All three metrics give us an idea of how well the model performs. However, they all emphasise different aspects of performance. For example, it might be hard to reach a perfectly accurate algorithm, but if we have a high enough recall, we can conclude that we got the most anomalous cases, which might be good enough for an auditor. The different metrics give us a broader view of what the independent variables do when adjusted.

IV Results

This section provides the results of the scenarios described in Section III. For every scenario, we will describe what has been done, the performance of the DECLARE constraints individually, and the performance of the data programming model. Scenarios one, two, and three will be provided with graphs of the output. The following naming will be used when referring to constraints: "constraint_id.scenario.variant_id". The constraint ID refers to the first column in Table 2. Since constraints can sometimes be modified in multiple manners, a variant ID shows the results for each manner separately. The results are summarised in Appendix A. The results will be discussed in Section V.

A Scenario 0: The baseline

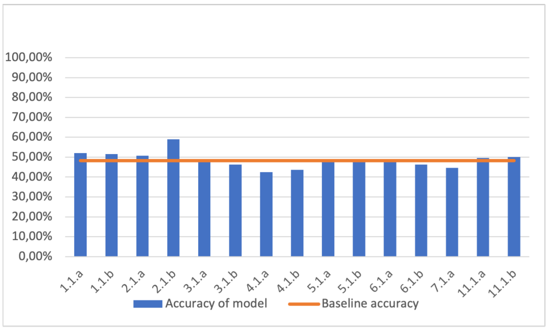
To establish a baseline to compare the different scenarios to, we will first use the experiment model as the set of constraints for the data programming model. The constraints from the experiment model (Figure 3) result in an accuracy of 48,2%, recall of 21,2%, and precision of 46,09%. The accuracy for each constraint individually can be found in Table 2. These values will be the baseline for our experiment. We can see how strengthening or weakening constraints influences the performance of our data programming model and the performance of the constraint.

ID	Activity A	Activity B	Relationship Type	Accuracy
1	Create PR	Approve PR	Responded existence(A, B)	50,7%
2	Approve PR	Create PO	Precedence(A, B)	46,2%
3	Create PO	Sign	Response(A, B)	50,0%
4	Sign	receive goods	Precedence(A, B)	51,3%
5	receive goods	Receive invoice	Alternate response(A, B)	36,0%
6	Receive invoice	Pay	Precedence(A, B)	50,1%
7	Sign	Pay	Co-existence(A, B)	54,3%
8	Create PR	/	Exactly(A, 1)	46,1%
9	Approve PR	/	Exactly(A, 1)	47,7%
10	Create PO	/	Exactly(A, 1)	81,2%
11	Sign	/	Existence(A, 1)	52,4%

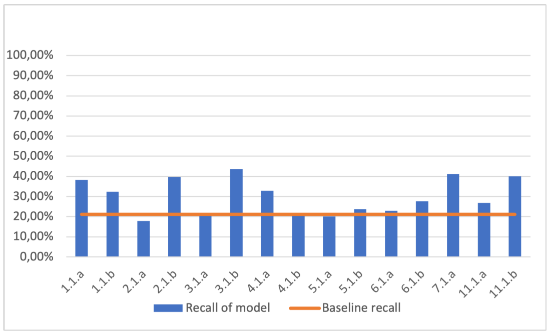
Table 2: DECLARE constraints of the baseline experiment model along with their accuracy

B Scenario 1: Strengthen one rule at a time

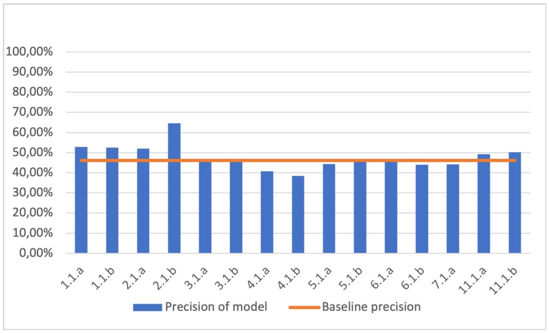
In this scenario, we strengthen every constraint individually by one level while keeping the other constraints constant. Figure 4a shows that the accuracy of the data programming model does not change significantly when strengthening a constraint. The overall mean shift in accuracy of the data programming model is 0,31%. Figure 4b shows that 11 out of 15 strengthened constraints increased recall of the data programming model. This with a mean of 8,79%. The shift in precision (Figure 4c) has a mean of 1,73% when compared to the baseline. Figure 4d shows how the strengthened constraints' accuracy compares to its baseline version. Constraint 4.1.b grew significantly (+27,80%) compared to its baseline value. Overall, the accuracy of the strengthened constraints changed only with a mean of 0,72% when compared to the constraints' baseline values. Table 4 shows the exact performance metrics for each strengthened constraint.



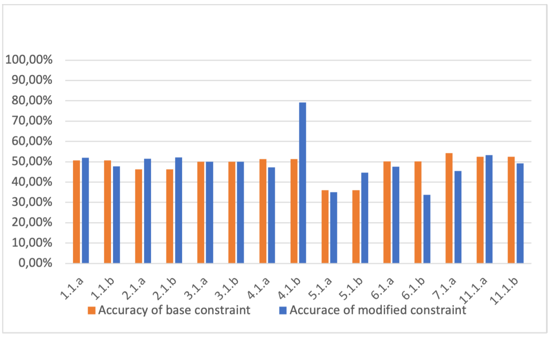
(a) Accuracy of the data programming model compared to the baseline after strengthening a constraint



(b) Recall of the data programming model compared to the baseline after strengthening a constraint



(c) Precision of the data programming model compared to the baseline after strengthening a constraint

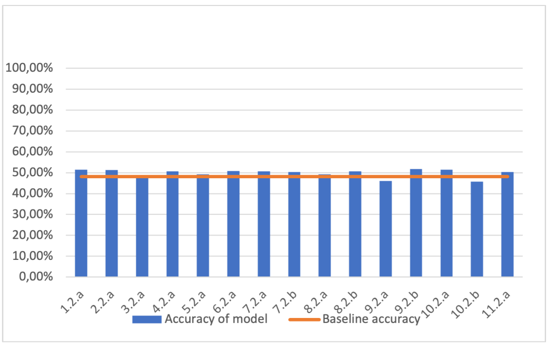


(d) Accuracy of the strengthened constraint compared to its baseline value

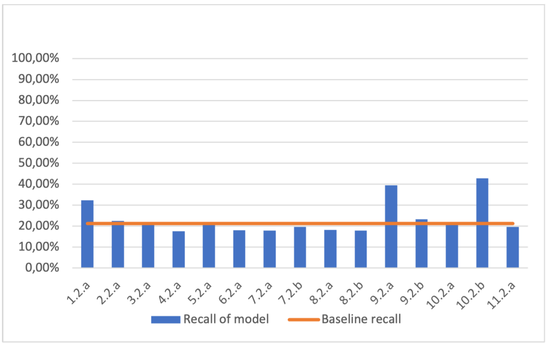
Figure 4: Results of strengthening constraints individually (Scenario 1)

C Scenario 2: Weaken one constraint at a time

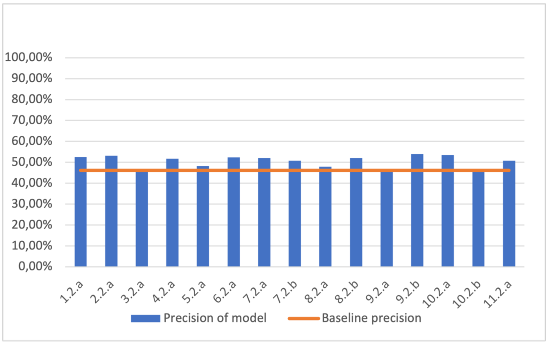
Instead of strengthening, we will weaken every constraint individually by one level. The accuracy of the data programming model (Figure 5a) increased with a mean of 1,65%. The recall (Figure 5b) changed with 2,33%. However, weakened constraints 1.2.a, 9.2.a and 10.2b caused an increase in recall of over 10%. Respectively, they increased the recall by 11,20%, 18,20% and 21,60%. The mean shift in precision (Figure 5c) is 4,29%. Figure 5d visualises how the weakened constraint performs in accuracy compared to its baseline value. On average, the accuracy of constraints dropped by 0,80% when weakened. However, constraints 6.2.a, 10.2.a and 10.2.b differ more, respectively 14,10%, -26,10% and -19,60%. Important to know is that constraints 1.2.a and 11.2.a did not result in an accuracy of 0%. These constraints were already at their weakest. The only possibility to weaken them was to remove them. Table 5 shows the exact performance metrics for each weakened constraint.



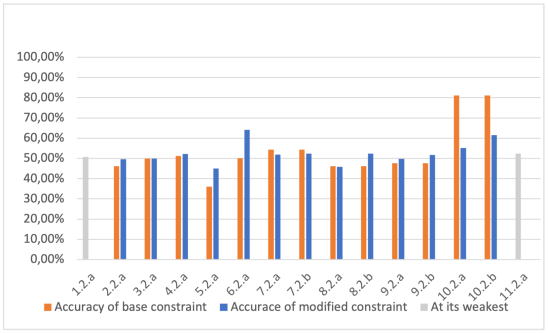
(a) Accuracy of the data programming model compared to the baseline after weakening a constraint



(b) Recall of the data programming model compared to the baseline after weakening a constraint



(c) Precision of the data programming model compared to the baseline after weakening a constraint



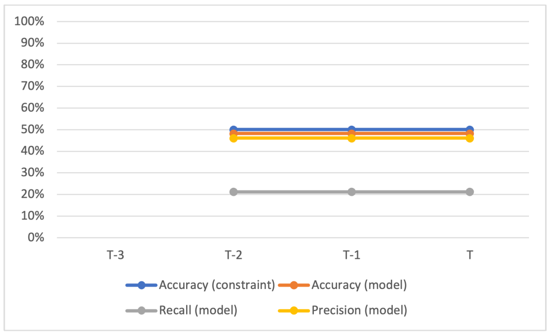
(d) Accuracy of the weakened constraint compared to its baseline value

Figure 5: Results of weakening constraints individually (Scenario 2)

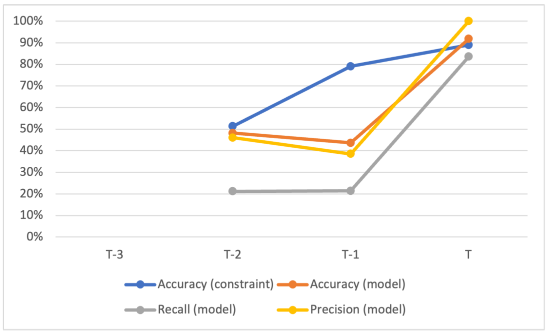
D Scenario 3: Change a constraints' baseline value towards auditor model value

In this scenario, we modify constraints from their baseline value (Table 2) toward the target value (T) found in the auditor model (Table 3). This will be done by strengthening or weakening the constraints to shift the relationship type closer to our target value. While shifting, we will keep all other constraints constant. However, since only four constraints are present in both the experiment and the auditor process model, this scenario was only applied to these four constraints.

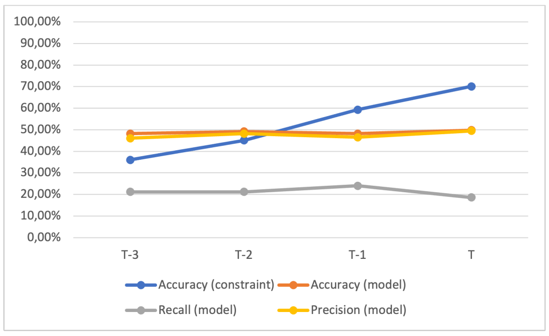
Figure 6 below shows the performance of the constraints and the data programming model when modifying the relationship type between activities from baseline value towards the auditor process model value (referred to as T in the figures below). Modifying constraint three did not change the performance for both the data programming model and the constraint. Constraint four's accuracy grew by 27,8% after modifying the baseline constraint once. The second modification caused it to increase the constraint's accuracy by another 9,9%. However, the performance of the data programming model only increased after the second modification. Modifying constraint five allowed it to increase its accuracy up to 70,1% gradually. However, the data programming model performance did remain roughly the same throughout all modifications. Constraint six's accuracy fluctuated throughout the modifications, but the data programming model responded the same as the modifications on constraint five. The exact values of each constraint modification can be found in Table 6.



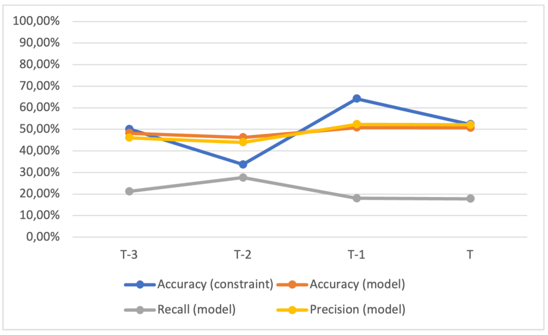
(a) Change in performance when modifying constraint 3



(b) Change in performance when modifying constraint 4



(c) Change in performance when modifying constraint 5



(d) Change in performance when modifying constraint 6

Figure 6: Result of gradually changing a constraint's baseline value (leftmost data point) towards the auditor model value (T) (Scenario 3)

E Scenario 4: The auditor model

In this scenario, we insert the DECLARE constraints of the auditor model (Figure 2) into the data programming model. We do this to check the capabilities of data programming. Since we used the auditor model to generate the data, we know it correctly represents allowed (exceptional) behaviour.

When using the auditor model, the data programming model has an accuracy score of 96,4%, recall of 92,8%, and precision of 100%. The accuracy of the individual DECLARE constraints can be found in Table 3.

Activity A	Activity B	Relationship Type	Accuracy
Create PO	Sign	Chain Response	50%
Sign	Receive goods	Response	89%
Receive goods	Receive invoice	Co-existence	70,1%
Receive goods	Pay	Responded existence	50%
Receive invoice	Pay	Responded existence	50%
Sign	Receive invoice	Response	88,9%
Create PO	/	Exactly 1	81,2%

Table 3: DECLARE constraints of Auditor model + accuracy per constraint

V Discussion

First, both weakening and strengthening DECLARE constraints did not result in a shift of data programming model accuracy, nor did it shift in a particular direction. In both scenarios, the model's accuracy increased on average, but the magnitude is too small to say that the model's accuracy correlated directly to the constraint modification. The data programming model compensates for the change in the accuracy of the modified constraint. However, this compensation happens when the constraint's accuracy increases or decreases. The model shifts the weights of the different constraints so that the impact on the model's performance is minimal. If the model only correlated to the altered constraint, then the change in performance would be much more noticeable. This can be seen in 6.2.a, 10.2.a and 10.2.b. Here, the constraint accuracy differs significantly from its baseline value, but the accuracy of the data programming model shifts only minimally. The weights are shifted to minimise the impact of losing a highly accurate constraint or prevent itself from overestimating a constraint with a jump in accuracy. This phenomenon can also be seen when we remove constraint three in the third scenario. This constraint has an individual accuracy of 50%. When the data programming model assigns weight to this constraint, it inserts some randomness in the model. If we remove this constraint, the data programming model will perform better because the weights will differ without randomness.

The recall does differ between the different scenarios. For scenario one, the recall improves in most cases with a mean value of 8,79%. This can be explained since strengthened rules will specify their constraint more explicitly. If the constraint gains accuracy, the data programming model will assign more weight to this constraint resulting in the higher recall due to the increased number of identified true positives. However, if the constraint is too specific, it will become less accurate; hence, other more generalistic constraints will gain weight. Since both scenarios apply, strengthening constraints does not directly cause an increase in recall. It does cause a shift in weights the data programming model assigns. However, it is unknown how these weights are assigned. In scenario two, the recall does not shift that much. However, on some occasions, it did result in a higher recall. The reason behind this needs to be clarified. One of the possibilities is that the weakened constraint ensured that another constraint grew in importance, receiving more weight from the model. When this is the case, the weakening of the constraint is not directly responsible for the increase. The shift of weights and, thus, the importance of the constraints shift the performance.

The precision of the data programming model follows the same trend as the accuracy but is amplified. This finding seems to be coincidental. No scientific source states that precision and accuracy are correlated. Mathematically, we find that when the data programming model can detect more true negatives, the accuracy increases while precision remains the same. However, further research is needed to confirm that this was coincidental. When strengthening constraints, we expect the precision to rise. However, in scenario one, the precision of the model increased on average less than when constraints were weakened. There is no explanation as to why this is the case. The only recommendation is to look in-depth at what happens with the weight distributions. This might give us a better understanding of why the data programming model behaves like this.

Furthermore, the individual constraints' accuracies do not change in any specific direction for scenarios one and two. For both scenarios, the constraints accuracies changed on average with less than 1%. Hence, we cannot state that strengthening or weakening has any predictable influence on the constraint. This can also be seen in scenario three. For all constraints tested, the constraint's accuracy either remained the same (constraint three), increased logarithmic (constraint four), increased linearly (constraint five) or fluctuated (constraint six). They all were strengthened or weakened to become the auditor process model value but behaved differently. Remarkable is the sudden increase of accuracy for constraint 4.1.b. There is no evidence that strengthening from a precedence type constraint to a succession type constraint increases accuracy since 6.1.b did the opposite and decreased by nearly 20%. It might be that the activities described by constraint 4.1.b are a good pair to label the cases, but the base value was too generalistic and hence had too many mistakes.

A fifth observation can be seen in scenario three. More specifically, in the case of constraint four during the third scenario. At first, the constraint grew in accuracy significantly but hurt the performance of the data programming model. However, slightly increasing the constraint's accuracy doubled the model's performance. A study by Laghmouch M. et al. (2020) explains this through the quality of the constraints. They claim only a small amount of medium- to high-quality rules are needed to make acceptable accurate predictions. After the first adaptation, the constraint's quality turned from low to medium. After the last adaptation, the medium-quality constraint turned into a high-quality constraint. The same reasoning applies to constraints four and five in the third scenario. These constraints never became high-quality rules. Hence the data programming model did not assign enough weight to the constraint to make it decisive. They did become medium-quality rules. When we changed constraints four and five to their medium quality variant together, the model's performance did increase. The study by Laghmouch et al. (2020) achieved reasonable performance with four to six medium- or high-quality rules. In this experiment, we only needed two high-quality constraints or one high-quality with two medium-quality rules. Even though both experiments were conducted in a procure-to-pay process with the same auditor model, fewer constraints were needed. The difference might be explainable due to two reasons. First, the auditor model is the same in both experiments, but the data used was different. A second explanation is that in the study of Laghmouch et al. (2020), the data programming model was allowed to abstain from labelling. This was done by feeding the model with constraints that either labelled a case if the case passed the constraint or abstained if it did not. In this experiment, we did not allow the model to abstain by using constraints that labelled cases either way. If the case passed a test, it was considered allowed behaviour; if it did not, it was considered anomalous.

VI Future work

First, the results in this experiment are specific to the procure-to-pay process, but the results might differ for other processes and data sets. We suggest that future research experiments on constraint modification with different processes and different datasets (i.e., unbalanced datasets). This is to confirm two hypotheses and extrapolate the findings to other processes: individual constraint modifications do not significantly impact data programming performance, and strengthening or weakening constraints does not change the constraint's accuracy in a specific direction.

There is one exception. This experiment shows a correlation between strengthening constraints and increased recall performance of the data programming model. However, strengthening constraints does not cause this increase directly. We suggest looking into the shift of weights instead of the accuracy of the constraint to understand what happens when constraints are strengthened.

During the experiment, we found multiple sets of constraints that resulted in an accuracy of 90% for the data programming model. Therefore, we can confirm the study of Laghmouch et al. (2020). As stated in the discussion, we achieved this with fewer constraints. We propose to research whether the way we formulate DECLARE constraints does impact the number of constraints needed. As a start, research can explore the difference in the performance of a data programming model between one that uses constraints that allow a model to abstain from labelling and one that uses constraints that enforce the model always to label.

VII Conclusion

This paper explored the relationship between constraint modification and data programming performance in a business process context.

Overall, we did not find a correlation between modifying constraints and the performance of the data programming model. The only way to improve the model's accuracy is to obtain enough medium- or high-quality constraints. This confirms the findings of Laghmouch et al. (2020). However, by using the constraint hierarchy of Schunselaar et al. (2012), we cannot modify constraints into qualitative constraints in a controlled manner. One of the reasons is that we cannot show a correlation between strengthening/weakening constraints and the change in the accuracy of the constraint. More specifically, strengthening or weakening constraints does influence the accuracy of the constraint but not in a predictable magnitude or direction. There is, however, a correlation between strengthening constraints and the recall of the data programming method.

From these findings, we propose three recommendations for future research. First, we cannot extrapolate our findings for all business processes since we experimented with only the procure-to-pay process. Therefore we propose future research to review our findings using different business processes. A second research recommendation might be the influence of allowing a data programming model to be indecisive and thus not label a case. Since our experiment forced the model to make a binary classification choice, this might have influenced the number of constraints needed to achieve reasonable performance.

At last, the correlation between strengthening constraints and the recall of the data programming model needs to be further investigated. We propose looking at the shift of weight that occurs when strengthening to find an explanation for this finding.

References

Alpaydin, E. (2010). *An introduction to machine learning.* MIT press.

Borning, A., Maher, M. J., Martindale, A., & Wilson, M. (1989). Constraint hierarchies and logic programming. In *Iclp* (Vol. 89, pp. 149–164).

Cugola, G. (1998). Tolerating deviations in process support systems via flexible enactment of process models. *IEEE Transactions on Software Engineering*, *24*(11), 982-1001. doi: 10.1109/32.730546

Cunningham, P., Cord, M., & Delany, S. J. (2008). Supervised learning. *Machine learning techniques for multimedia: case studies on organization and retrieval*, 21–49.

Dickey, G., Blanke, S., & Seaton, L. (2019). Machine learning in auditing. *The CPA Journal*, *89*(6), 16–21.

Hossin, M., & M.N, S. (2015, 03). A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining Knowledge Management Process*, *5*, 01-11. doi: 10.5121/ijdkp.2015.5201

Jans, M., & Laghmouch, M. (2023). Process mining for detailed process analysis. In E. Berghout, R. Fijneman, L. Hendriks, M. de Boer, & B.-J. Butijn (Eds.), *Advanced digital auditing: Theory and practice of auditing complex information systems and technologies* (pp. 237–256). Cham: Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-031-11089-4_9 doi: 10.1007/978-3-031-11089-4_9

Laghmouch, M., Jans, M., & Depaire, B. (2020). Classifying process deviations with weak supervision [Proceedings Paper]. In *2020 2nd international conference on process mining (icpm 2020)* (pp. 89–96). 10662 LOS VAQUEROS CIRCLE, PO BOX 3014, LOS ALAMITOS, CA 90720-1264 USA: IEEE COMPUTER SOC. (2nd International Conference on Process Mining (ICPM)) doi: 10.1109/ICPM49681.2020.00023

Nonnenmacher, J., & Marx Gómez, J. (2021, 01). Unsupervised anomaly detection for internal auditing: Literature review and research agenda. *The International Journal of Digital Accounting Research*, 1-22. doi: 10.4192/1577-8517-v21_1

Pesic, M., Schonenberg, H., & van der Aalst, W. M. (2007). Declare: Full support for loosely-structured processes. In *11th ieee international enterprise distributed object computing conference (edoc 2007)* (p. 287-287). doi: 10.1109/EDOC.2007.14

Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., & Reijers, H. A. (2011). *Imperative versus Declarative Process Modeling Languages: An Empirical Investigation.* Springer Science+Business Media. Retrieved from `https://doi.org/10.1007/978-3-642-28108-2_37` doi: 10.1007/978-3-642-28108-2\{_}37

Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., & Ré, C. (2017, November). Snorkel: Rapid training data creation with weak supervision. *Proceedings VLDB Endowment*, *11*(3), 269–282.

Ratner, A., De Sa, C. M., Wu, S., Selsam, D., & Ré, C. (2016). Data programming: Creating large training sets, quickly. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 29). Curran Associates, Inc.

Ratner, A., Varma, P., Hancock, B., & other members of Hazy Lab. (2019, 3). *Weak Supervision: A New Programming Paradigm for Machine Learning.* Retrieved from `https://ai.stanford.edu/blog/weak-supervision/`

Schunselaar, D. M. M., Maggi, F. M., & Sidorova, N. (2012). Patterns for a log-based strengthening of declarative compliance models. In J. Derrick, S. Gnesi, D. Latella, & H. Treharne (Eds.), *Integrated formal methods* (pp. 327–342). Berlin, Heidelberg: Springer Berlin Heidelberg.

Swinnen, J., Depaire, B., Jans, M., & Vanhoof, K. (2011, 08). A process deviation analysis – a case study. In (Vol. 99, p. 87-98). doi: 10.1007/978-3-642-28108-2_8

Vakili, M., Ghamsari, M., & Rezaei, M. (2020). Performance analysis and comparison of machine and deep learning algorithms for iot data classification. *CoRR*, *abs/2001.09636*. Retrieved from `https://arxiv.org/abs/2001.09636`

Zuhaira, B., Ahmad, N., Saba, T., Haseeb, J., Malik, S. U. R., Manzoor, U., ... Anjum, A. (2017). Identifying deviations in software processes. *IEEE Access*, *5*, 20319-20332. doi: 10.1109/ACCESS.2017.2757954

A Numeric results of the different scenarios

Var.	Activity A	Activity B	Base Relationship Type	New Relationship Type	Acc. of New Constraint	Accuracy Model	Recall Model	Precision Model
1.1.a	Create PR	Approve PR	Responded existence(A, B)	Co existence(A, B)	52%	52%	38,2%	52,76%
1.1.b	Create PR	Approve PR	Responded existence(A, B)	Response(A, B)	47,7%	51,5%	32,4%	52,43%
2.1.a	Approve PR	Create PO	Precedence(A, B)	Alternate precedence(A, B)	51,5%	50,7%	17,8%	52,05%
2.1.b	Approve PR	Create PO	Precedence(A, B)	Succession(A, B)	52,1%	59%	39,8%	64,61%
3.1.a	Create PO	Sign	Response(A, B)	Alternate response(A, B)	50%	48,2%	21,2%	46,09%
3.1.b	Create PO	Sign	Response(A, B)	Succession(A, B)	50%	46,2%	43,6%	45,99%
4.1.a	Sign	receive goods	Precedence(A, B)	Alternate precedence(A, B)	47,2%	42,5%	32,8%	40,69%
4.1.b	Sign	receive goods	Precedence(A, B)	Succession(A, B)	79,1%	43,6%	21,4%	38,49%
5.1.a	receive goods	Receive invoice	Alternate response(A, B)	Chain response(A, B)	35,1%	47,4%	20,2%	44,3%
5.1.b	receive goods	Receive invoice	Alternate response(A, B)	Alternate Succession(A, B)	44,6%	48%	23,8%	46,12%
6.1.a	Receive invoice	Pay	Precedence(A, B)	Alternate precedence(A, B)	47,6%	48,1%	23%	46,18%
6.1.b	Receive invoice	Pay	Precedence(A, B)	Succession(A, B)	33,7%	46,2%	27,6%	43,95%
7.1.a	Sign	Pay	Co-existence(A, B)	Succession(A, B)	45,5%	44,6%	41,2%	44,21%
8.1.a	Create PR	/	Exactly(A, 1)	N/A				
9.1.a	Approve PR	/	Exactly(A, 1)	N/A				
10.1.a	Create PO	/	Exactly(A, 1)	N/A				
11.1.a	Sign	/	Existence(A, 1)	Exactly(A, 1)	53,2%	49,6%	26,8%	49,26%
11.1.b	Sign	/	Existence(A, 1)	Existence(A, 2)	49,2%	50,1%	40%	50,13%

Table 4: Results of scenario 1

Var.	Activity A	Activity B	Base Relationship Type	New Relationship Type	Acc. of New Constraint	Accuracy Model	Recall Model	Precision Model
1.2.a	Create PR	Approve PR	Responded existence(A, B)	N/A	/	51,5%	32,4%	52,43%
2.2.a	Approve PR	Create PO	Precedence(A, B)	Responded existence(B, A)	49,6%	51,3%	22,4%	53,08%
3.2.a	Create PO	Sign	Response(A, B)	Responded existence(A, B)	50%	48,2%	21,2%	46,09%
4.2.a	Sign	receive goods	Precedence(A, B)	Responded existence(B, A)	52,2%	50,6%	17,6%	51,76%
5.2.a	receive goods	Receive invoice	Alternate response(A, B)	Response(A, B)	45,1%	49,2%	21,2%	48,18%
6.2.a	Receive invoice	Pay	Precedence(A, B)	Responded existence(B, A)	64,2%	50,8%	18%	52,33%
7.2.a	Sign	Pay	Co-existence(A, B)	Responded existence(A, B)	51,9%	50,7%	17,8%	52,05%
7.2.b	Sign	Pay	Co-existence(A, B)	Responded existence(B, A)	52,4%	50,3%	19,6%	50,78%
8.2.a	Create PR	/	Exactly(A, 1)	Absence(A, 2)	45,9%	49,2%	18,2%	47,89%
8.2.b	Create PR	/	Exactly(A, 1)	Existence(A, 1)	52,3%	50,7%	17,8%	52,05%
9.2.a	Approve PR	/	Exactly(A, 1)	Absence(A, 2)	49,8%	46%	39,4%	45,39%
9.2.b	Approve PR	/	Exactly(A, 1)	Existence(A, 1)	51,7%	51,7%	23,2%	53,95%
10.2.a	Create PO	/	Exactly(A, 1)	Absence(A, 2)	55,1%	51,4%	21,8%	53,43%
10.2.b	Create PO	/	Exactly(A, 1)	Existence(A, 1)	61,6%	45,8%	42,8%	45,53%
11.2.a	Sign	/	Existence(A, 1)	N/A	/	50,3%	19,6%	50,78%

Table 5: Results of scenario 2

Var.	**Activity A**	**Activity B**	**Base Relationship Type**	**Target relationship type**	**T-3**				**T-2**				**T-1**				**T**			
					const.	Acc	Rec	Pre	const.	Acc	Rec	Pre	const.	Acc	Rec	Pre	const.	Acc	Rec	Pre
3.3.a	Create PO	Sign	Response(A, B)	Chain Response(A, B)					50%	48,2%	21,2%	46,09%	50%	48,2%	21,2%	46,09%	50%	48,2%	21,2%	46,09%
4.3.a	Sign	Receive goods	Precedence(A, B)	Response(A, B)					51,3%	48,2%	21,2%	46,09%	79,1%	43,6%	21,4%	38,49%	89%	91,8%	83,6%	100%
5.3.a	Receive goods	Receive invoice	Alternate response(A, B)	Co-existence(A, B)	36%	48,2%	21,2%	46,09%	45,1%	49,2%	21,2%	48,18%	59,3%	48,2%	24%	46,51%	70,1%	49,8%	18,6%	49,47%
6.3.a	Receive invoice	Pay	Precedence(A, B)	Responded existence(A, B)	50,1%	48,2%	21,2%	46,09%	33,7%	46,2%	27,6%	43,95%	64,2%	50,8%	18%	52,33%	50%	50,7%	17,8%	52,05%

Table 6: Results of scenario 3