Process Mining on Event Graph Databases
Exploratory Case Study on an ITIL Incident Analysis

Türkyılmaz, Pinar

Award date:
2020

Department of Mathematics and Computer Science
Process Analytics Charter

Process Mining on Event Graph Databases

Exploratory Case Study on an ITIL Incident Analysis

Master Thesis

Pinar Turkyilmaz

Supervisors:
Dr. Dirk Fahland

15-08-2020

Abstract

Graph databases allow storing the multi-dimensional event data in a format that makes it easy to follow the traces. Property Graphs can store case identifiers in nodes. Moreover, the information between each event, between each case and between cases and events can be stored under relationships. Therefore, behavioral sequential information of the events, which are traces, can be represented as paths. Hence, the results become more reliable and easily interpretable. There are two different schemas previously implemented for event logs in property graphs, however there is no process mining analysis executed on a graph structure. Therefore, a research is essential on a process mining analysis on graph databases. In this thesis, an ITIL process is used for an impact analysis of the changes to IT components. We discuss the requirements of a data model and the structure should satisfy. Moreover, we propose a new schema where the analysis will be conducted on and we compare the results with the classical process mining techniques gives. Finally, we visualize our findings to grasp the efficiency of graph structure and gain more elaborated insights than a classical process mining analysis gives.

Keywords: Process Mining, Graph Databases, Data Analysis, Labeled Property Graph, Multi-Dimensional Event Data, ITIL

Preface

This master thesis concludes my studies for the Master's program Big Data Management and Analytics which was held in Universite Libre de Bruxelles, Universitat Politecnica de Catalunya and Eindhoven University of Technology.

First of all, I would like thank to my supervisor Dirk Fahland for his excellent guidance and valuable feedbacks throughout this project. All of them were very well appreciated.

Furthermore, I want to thank to my parents, Nuket and Nevzat Turkyilmaz for their great compassion and support throughout my life. Thank you for allowing me to grow in the way I want. I also would like to thank to my brother, Ogun Turkyilmaz for being there whenever I need and for all the joyful trips we had, not only in these two years but also in the past.

Last but not least, I would like to thank to all of my friends who I met during my entire studies.

Contents

Chapter 1

Introduction

In this Chapter we first provide the overall context of this thesis in Section 1.1. Then we present the conducted research related to event graph databases, process mining on relational databases, and graph databases in Section 1.2. Section 1.3 and Section 1.4 defines the research question and our methodology we used to conduct our analysis. Finally, Section 1.5 gives the results we have.

1.1 Thesis Context

Event logs as collection of events are the starting point of process mining. The Process mining techniques are assuming to use sequential recorded event logs [15]. Sequential event logs keep one case identifier for each event. Therefore, the event data can be stored in the event log as one sequence of events per case. These sequences can be queried for the behavioral properties.

However, many processes involve multiple interrelated entities which gives us the multi-dimensional event data. In multi-dimensional event data, each event is directly or indirectly related with multiple case identifiers [7]. When one entity occur in multiple cases, we would not be able to represent this information with sequential event logs. Therefore, it would result in information loss.

Graph databases can overcome this problem with linking events to entities using the edges and model directly-follow events per entities, therefore, they allow to represent event data with multiple case identifiers. Event and case identifiers can be stored in nodes and the information between each event, between each case and between cases and events can be stored under relationships. It allows to follow the traces for each cases very easily and behavioral sequential information of the events, which are traces, can be

represented as paths.

Esser [6] and Lander [10] proposed two different approaches to create graph based event logs. Existing process mining techniques use sequential event logs not graphs. Therefore, it is very essential to understand how such an analysis can be conducted.

1.2 State of the Art

There are event graph data implementations in two previous master's thesis by Esser [6] and Leander [10] and one paper by Esser and Fahland [14].

Moreover, there are studies for process mining in on the relational databases by Dijkman, R., Gao, J., Syamsiyah, A. et al [5] who stated that it is hard to work on Case base on relational databases and proposed the optimized queries for such analysis.

There is another study by Romero [13] which mainly concentrated on the definition of spatial and temporal settings on graph databases.

Mendelzon and Wood [12] also worked on algorithms on graph databases to obtain a path satisfying a given regular expression.

The reports on BPIC14 uses different process mining techniques and tools to mainly predict the impact of Change on ITIL process [1].

To conclude, there are studies on graph databases and process mining on relational databases. Moreover, the reports presented in BPIC14 propose analysis on ITIL dataset. There are two event graph data implemented by Esser and Leander. However, there is no any process mining analysis executed on the schemas and a further investigation is necessary whether to result it is possible to execute such analysis on graph event data.

1.3 Research Question and Methodology

As clearly stated in the 1.2, there is not a process mining analysis on the graph databases. There is a need to explore how an analysis can be executed. In this report we will discuss the process mining analysis on graph databases. More particularly "Is it possible to conduct process mining analysis on graph databases?". The sub-questions are as following:

- Which data model is appropriate for process mining?
- Which queries and analysis steps are necessary to execute an impact analysis of the changes which IT components experiences on graph data?

- Is it possible to obtain the same results with the classical process mining techniques and tools?

1.4 Methodology

We execute a case study on ITIL IT Management process data. ITIL data is a multi-dimensional and available in both data schemas. Furthermore, there are 13 reports presented in BPIC14. Many of them worked on the prediction model to predict the impact of Changes. They used different techniques and process mining tools [1]. We chose a paper by Hanser. They conducted only process mining analysis. However, in any of these analyses, graph databases has never been used.

First, we discuss the requirements of a schema should fulfill to conduct a process mining analysis. The obligations were checked in terms of presenting a proper DF relation between events, a proper activity definition of the events and the efficiency of the queries that should be written in the particular schema.

Secondly, we processed the data to increase data quality. There were many typical problems that could be seen in any data. We show the efficient ways correct the graph based event logs without any information loss.

Thirdly, we prepared the data for the impact analysis of Changes. We aggregate the events into one event but keep the DF relation between them. We order the events based on the last event timestamp of the entity. Then, we calculated the Interactions and Incidents between each Change and call them as previous/post Incidents, and previous/post Interactions for each Change.

Furthermore, we created new schema with the aggregated events for each Configuration Items and call it *Analysis Schema*. We conducted the impact analysis on Analysis Schema. if a Changes is followed by less Interactions/Incidents than previous we call it as Positive Change. However, whenever a Changes is followed by more Interactions/Incidents than previous we call it as Negative Change.

Fig. 1.1 gives how the implementation of analysis executed. As an example we created Fig. 1.1 for detailing the analysis idea. there are 2 CIs, CI1 and CI2, and the sequence of Changes and Incidents that they both experience. CHG3 is effecting both of them. On CI2, we say that it has a positive impact because the following Incidents until next Change are less then the Incidents from the previous Change. Meanwhile, on CI1, CHG3 has a negative impact because after the implementation, it increases the Incidents.

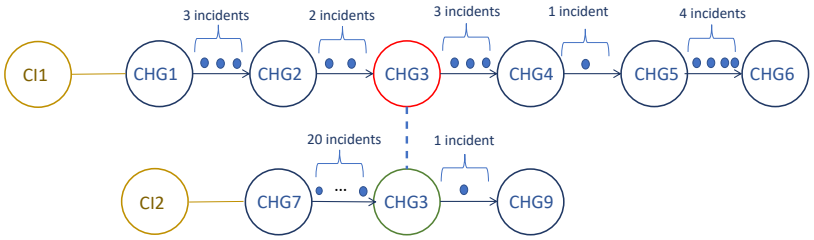


Figure 1.1: Idea of the Analysis. CHG3 has positive impact on CI2 and negative impact on CI1.

Then,we compared our analysis to an analysis where classical process mining techniques on sequential event logs were applied.

Last but not least, we create two different type of Visualization to elaborate our findings more.

We first create Visualization for the Configuration Items to see the impact of Changes which effect multiple Configuration Items. The outcome is the path of that IT Component is experiencing and same Changes are clustered based on the color and connected with a line. Then, another visualization is a scatter plot to see the impacts how a Configuration Item is mainly effected by a Change.

Fig. 1.2 presents more details the overview of the steps for the executed impact analysis.

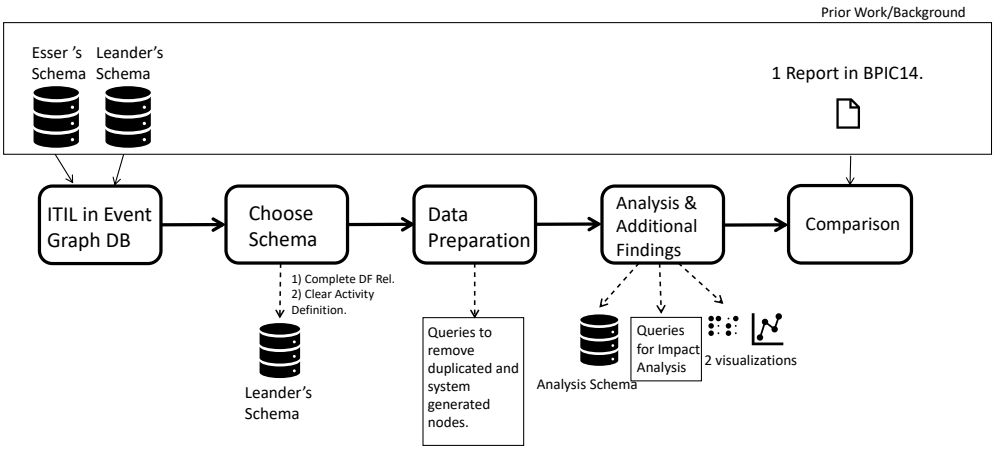


Figure 1.2: Overview of the steps through the analysis

1.5 Findings

We found that the data model should contain a complete Directly Follow relationships and clear activity definition for events. Esser's Schema proposes Events as belonging to multiple entities. This approach results as one Event can have multiple DF relation with different Events. Leander's schema provides more convenient solution as every Entity has its own Directly Followed Events. In terms of activity definition, Esser's schema does not offer a clear even description. However, Leander's schema has every necessary information. Furthermore, it is possible to obtain the results slower because of too many nodes to traverse. In Leander's schema the queries was taking much more time whereas Esser's schema was way faster. Details are provided in Chapter 3.

For conducting the analysis, first the data was checked whether it is suitable for the analysis. We found the methods to delete the duplicated and unnecessary information without ruining the database schema. Further details are in Chapter 4.

Section 4.4 provides details about aggregation of all event nodes of Entities into one Event node but store the relationship between them. Then we calculated the previous/post Incidents and Interactions between each Changes.

To be able to execute the impact analysis, we created a new schema, so-called analysis schema, by aggregating events of one Entity into one Event. We order order these Events based on last timestamp of events of each Entity. So, every IT components have the entities in order. Further details are in Section 4.5. Then, in we conducted the analysis and found the following results.

- Impact of Interaction: There are 996 Negative Changes. There are 1211 Positive Changes. 650 Changes are not followed by any Interactions. This can be because of the Change is at the end of the data or the Change occurred because of Problems not Interactions or the Change was a perfect implementation solved the problems. Overall, this step shows that Changes are very useful for Interactions. We also checked, how the Configuration Items are effected by the Changes. We found that there are 448 Configuration Items that has experiencing Interactions eventually followed by Changes. Meanwhile, 468 Configuration Items experience Changes eventually followed by Interactions. We can conclude as in the Configuration Item level, the effect of Changes can vary between the products. Therefore, there is not a strong correlation that we can rely on.

- Impact of Incident: There are 996 Negative Changes. There are 1271 Positive Changes. Meanwhile, 772 Changes which are not followed by any Incidents and it can be because of the Change is at the end of the data or the Change occurred because of Problems not Incidents or the Change was a perfect implementation solved the problems. Overall, this shows that Changes are very useful. We also checked, how the Configuration Items are effected by the Changes. We found that there are 532 Configuration Items that has experiencing Incidents eventually followed by Changes. 559 Configuration Items experience Changes eventually followed by Incidents. The effect of Changes can vary between the products. Therefore, there is not a correlation that we strongly can rely on.

We found some Changes are effecting multiple Configuration Items at the same time. Based on this, we developed a Visualization to analyse the impact of Changes for multiple Configuration Items. It shows that the effect of Changes is varying. One Change can have positive impact on one product, and negative impact on others. Furthermore, one IT Component's experience can differentiate between Incidents and Interactions. With the help of this visualization, we were able to identify the impact of Changes which effect

multiple Configuration Items in terms of different applications. Details are in Chapter 5

These results lead us to conclude the analysis as follows: We found that when considering changes per IT component in isolation that we obtained the same results. However, we found the individual changes which affected multiple IT components and have a mixed effect on different items. With process mining techniques this would not be identified. Graph-based structure helped us to see the whole path for an IT component. Therefore, we had more insights about the effects of Changes. Further details are in 6

We also provide the generalizable queries in Chapter 7 for typical problems that can occur in any dataset. We provide the general structure to make sure they can be used in different dataset.

Chapter 2

Background

This Chapter provides the definitions of related disciplines and related work. Section 2.1.1 gives the definition of process mining and identify where our analysis is placed In section 2.1.2 describe graph databases and labeled property graph model with information about Neo4j, a graph database management system, and query language for the property graph, Cypher. In Section 2.1.3, we give the details about ITIL processes and Section 2.1.4 introduces the questions in Business Process Intelligence Challenge 14, which our analysis is based on. In Section 2.1.5 we define CSV files as we use in our analysis. In section 2.2, we provide the existing research related Graph Databases, Process Mining on Databases, and Event Graph Data. Moreover, we also introduce one paper from BPIC14 and discuss their methodology.

2.1 Preliminaries

2.1.1 Process Mining

Wil van der Aalst et al. [15] describe process mining as a discipline of combination of business process management and data mining on the one hand, and process modelling and analysis on the other hand. The main purpose of process mining is to contribute to discover, monitor, and improve the business processes based on the information extracted from the event logs.

Process mining has many domains such as process discovery, conformance checking, social network/organizational mining, automated construction of simulation models, model extension, model repair, case prediction, and history-based recommendations.

This thesis is on an introductory bases where we discuss discuss the data understanding and feature extraction for a proper process mining analysis

on graph databases.

2.1.2 Graph Databases

A graph database is a database which stores and represents the data using relationships and nodes. It stored the entities as *nodes* and the way how these entities are connected are defined as *relationships*. In this project, we used labeled property graph contains nodes, which has key-value pairs, and relationship, which has name and always has start and end node [9].

The event logs in graph database has event and case identifiers which can be stored in nodes and the information between each event, between each case and between cases and events can be stored under relationships. Therefore, behavioral sequential information of the events, which are traces, can be represented as paths.

Neo4j and Cypher

Neo4j is an open-source, NoSQL database. Neo4j is referred to as a native graph database because it efficiently implements the property graph model down to the storage level [2]. Cypher is a declarative query language similar to SQL, but optimized for graphs. It allows to obtain all nodes and relationships that match a specific pattern [2]. Fig.2.1 shows a very simple graph pattern.

For this schema the path can be described as follows:

```
(Alice)-[:KNOWS]->(Mark)-[:KNOWS]->(Mary) and
   (Mary)-[:KNOWS]->(Alice)
```

We can also define a query to find mutual knows relation of Alice.

```
MATCH(a:User{Name:Alice})-[:KNOWS]->(b:User{Name:Mark})-
[:KNOWS]->(c:User{Name:Mary}), (c)-[:KNOWS]->(a)
Return b,c
```

MATCH starts with identifying the node as our starting point and describes the path that has to be followed. RETURN specifies which nodes should be returned.

There are also other important Clauses to look into.

- WHERE provides the restrictions for filtering the pattern.
- DELETE removes nodes, relationships, properties.

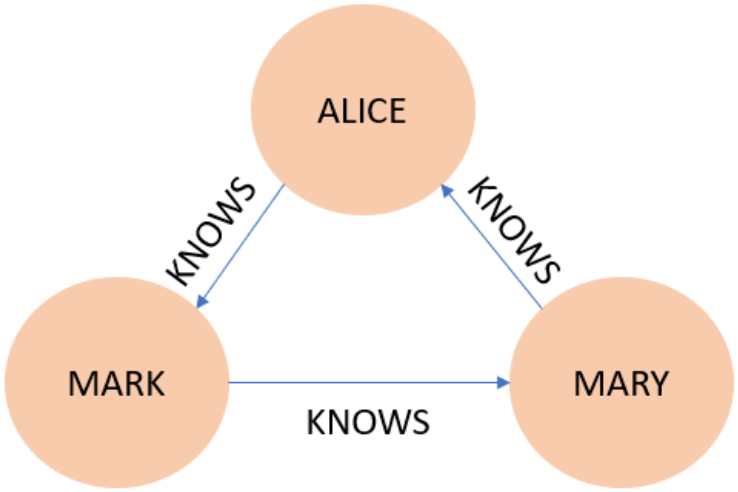


Figure 2.1: A simple graph pattern which describes Knows relation

- SET sets the property values.
- CREATE and CREATE UNIQUE creates nodes and relationships.
- MERGE ensures that no new object is created if that object exists.
- LOADCSV is the clause to Load the data from a CSV to Neo4j.
- WITH forwards the results to the next query.
- DISTINCT removes the duplications. Therefore, it ensures that the result only contains unique elements.
- FOREACH provides an action for each element in a list.
- ORDERBY provides the results ordered for a based on a related property.
- LIMIT ensures the result is returned in the limited amount.

2.1.3 ITIL process

ITIL is a collection of defined and published best practice processes for ITSM (IT Service Management.) [4]. Fig.2.2 shows that the ITIL framework contains seven process modules which situated between the business and technology.

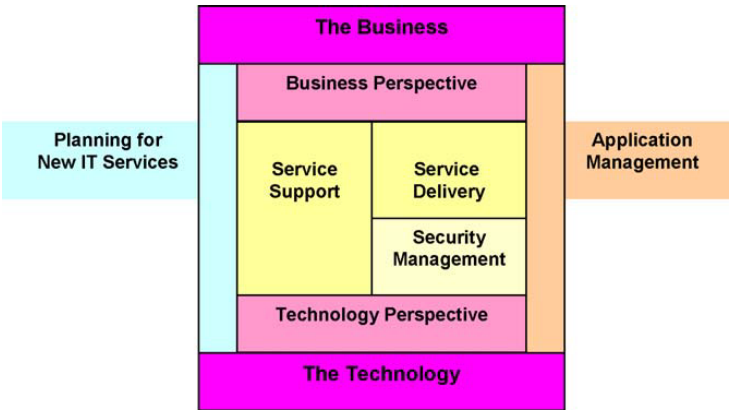


Figure 2.2: ITIL framework

As can be seen Service support is at the core of the process and we will focus on that. Fig. 2.3 shows the Service Support module.Service Support Module contains Incident Management, Problem Management, Change Management, Release Management, Configuration Management. We will mainly concentrated on Change Management.

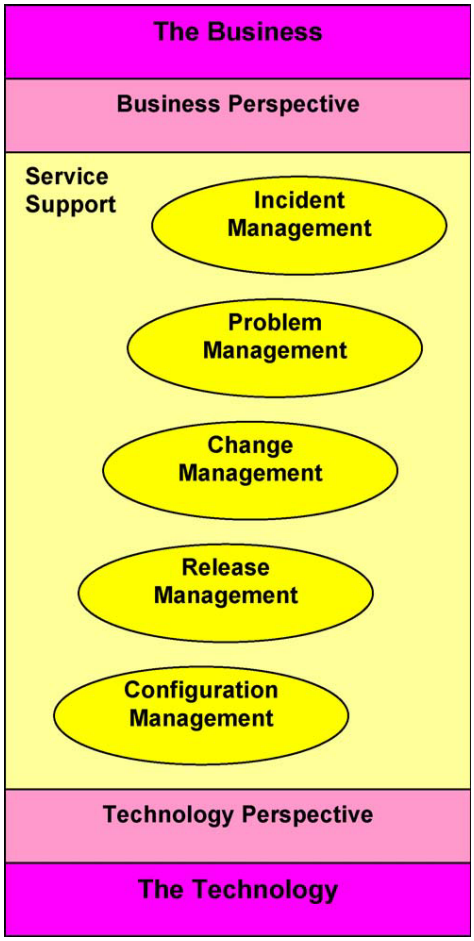


Figure 2.3: Service Support module

It also important to discuss about the way that Rabobank implemented ITIL process. Fig.2.4 illustrates the ITIL process in Rabobank. The general process can be summarized as a problem reported to the service desk may evolve from an Interaction to an Incident up to a Change request [3].

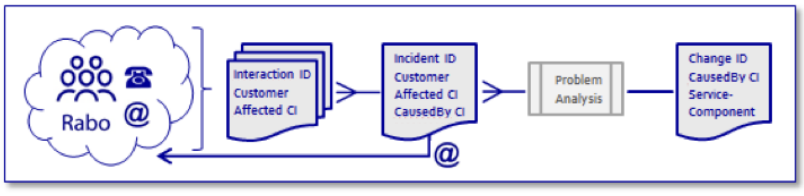


Figure 2.4: The ITIL process in Rabobank. Includes Interaction, Incident, Change.

Interaction Management Service Desk works with the customers directly. They mainly concern about the disruptions of ICT Services. They receive the calls/mails as an Interaction record and relates it to an Affected Configuration Item (CI). Service Desk can either solve the problem directly or create an an Incident-record to handover the service disruption to more technical group. If similar calls/mails are received by the Service Desk, multiple Interaction-records can be related to one Incident-record.

Incident Management An Incident record is prioritized and gets deadline based on estimated impact and urgency. It can be either solved or reassigned it more investigation is needed.

Change Management If particular disruptions reoccur many times, than it will lead to an analysis and improvement on the IT Component to prevent the disruption happen again.

2.1.4 Business Process Intelligence Challenge 14

The Business Process Intelligence 14 (BPIC14) was a challenge took place in 2014. The participants were asked to analyze an ITIL IT Management process data provided by Rabobank ICT [1]. The challenge was to create a predictive model to support Business Change Management. The main aim was to decrease the impact of the software releases on Service Desk and IT Operations. The sub-questions for this challenge were as following:

- Identification of impact pattern:Analysing the correlation between implementation of a Change and Service Desk and IT Operations. In more details the impact of Change on increased/decreased volume of Closed Interactions and/or increased/decreased volume of Closed Incidents.
- Parameters for every impact pattern:
    1. What is the average period to return to a steady state after a Change is implemented?
    2. What is the average increase/decrease of Closed Interactions once a new steady state is reached?
- Change in Average Steps to Resolution: Since project managers are expected to deliver the same or better service levels after each change implementation, Rabobank Group ICT is looking for confirmation that this challenge is indeed being met for all or many Service Components.

- Creativity challenge: Finding new insights on the provided data.

In this thesis, we will work on the first sub-question which is Identification of Impact Pattern.

2.1.5 Comma Seperated Values File- CSV

CSV file is the most prominent file format for moving tabular data. CSV data is a standard for exchanging and converting the data between different related applications. The key format of CSV data are as following [11]:

- CSV is a 2D (two dimensional) format constructed by rows and columns of data, each row containing multiple cells.
- CSV is a "text-based" format (i.e. a CSV file is a text file) which makes it flexible for processing with all types of textual applications (software).
- It is an easiest and very popular possible analytical format for tabular data.

Moreover, CSV data is structured into rows; individual row contains a number of information about data. Also, CSV files can have an optional header row and column names of the rows for each column.

2.2 Related Work

2.2.1 Graph Databases and Process Mining on Databases

Dijkman, R., Gao, J., Syamsiyah, A. et al [5] worked on process mining on relational databases. They defined a relational algebraic operator to extract the 'directly follows' relation from a log that is stored in a relational database, possibly distributed over multiple tables. The paper presents the relational algebraic operators which are selection, operation, and theta join for 'directly follows' relation. They mainly worked on the optimization of the queries and operational costs. As a future work, they stated that " Currently, it is hard to work with 'cases' in an SQL query, because a log table has a row for each event that happens rather than each case that completes.".

Another paper proposed a graph-based approach to modeling events and their interrelations. They mainly concentrated on the need of temporal and spatial settings to manage different levels of granularity [13].

Mendelzon and Wood [12] worked on finding the nodes in a labelled, directed graph which are connected by a simple path satisfying a given regular expression. They developed an algorithm to implement such queries.

2.2.2 Event Graph Data

Esser and Fahland [14] proposed a technique to store multi-dimensional event data in labeled property graphs. They also reported the queries on the graphs for structural and temporal properties together. Their main finding is that event data over multiple entities and identifiers with complex relationships can be stored in graph databases in a systematic way.

Esser [6] defined a property-graph templates to encode the fundamental event log concept. They tested the schema on six case studies on five different event logs datasets .

Leander [10] discussed how to automatically transform event data from a relational to a graph database with one-to-one, one-to-many and many-to-many relationships between events and case identifiers, such that the transformed data captures both structural and temporal relations in the data. They designed a graph data model that is able to capture these structural and temporal relations. Finally, they tested their structure on two datasets.

2.2.3 BPIC14 Report

There are 13 reports presented in BPIC14. Mainly the reports present building and analysing the prediction models by combining process mining and data mining techniques. However, we accept the report by Hanser [8] as the ground truth for our analysis, because they only used process mining techniques and tools to conduct the analysis. That would be more relevant with the topic of this thesis. They found that, there is not a strong correlation between Interactions/Incidents and Changes. For some Configuration Items Changes might have good impact but for the others a reverse impact can be observed. We compare our results with Hanser's report. We define their subquestions and methods for the analysis.

Subquestions

As mentioned in Section 2.1.4, the first question is the identification of impact patterns of Changes on Interactions and Incidents. Hanser divided this question into four subquestions.

- Subquestion 1.1: Is there in general a significant increase/decrease in

the number of interactions after implementing Changes compared to before implementing Changes?

- Subquestion 1.2: Is there a significant correlation between the increase in number of interactions being related to PRODUCTS with Changes, after the Changes have been implemented?
- Subquestion 1.3: Is there in general a significant increase/decrease in the number of incidents after implementing Changes compared to before implementing changes?
- Subquestion 1.4: Is there a significant correlation between the increase/decrease in number of incidents being related to PRODUCTS with Changes, after the Changes have been implemented

Method

This section introduces their preprocessing and method to answer each subquestion.

Subquestion 1.1 They filtered the data such as eliminating the Changes which does not have Actual End activity. It gives them 92% of the Changes and they examined the number of Interactions before and after implementing the Changes.

Subquestion 1.2 Hanser created one case for each product with Change and Interaction activities.

For the data filtering to find the Interactions before Change, they removed the cases which does not start with Interaction activity, and the cases which do not contain any Change activities. Also they trimmed the cases to contain all activities from the first Interaction events to the last Change event as can be seen on Fig.2.5.

For the data filtering for finding the Interactions after Change, they removed the cases which does not start with Interaction activity, and the cases which does not contain any Change activities. Also they trimmed the cases to contain all activities from the first Change events to the last Interaction event as can be seen on Fig.2.5.

Subquestion 1.3 They filtered the data such as eliminating the Changes which does not have Actual End activity. They examined the number of Incidents before and after implementing the Changes.

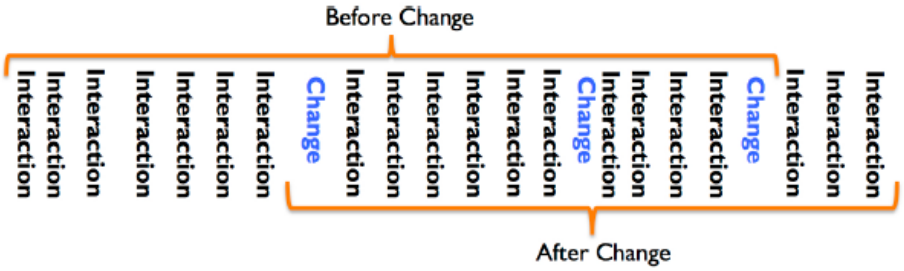


Figure 2.5: Hansen's method to calculate the Interactions Before and After Change.

Subquestion 1.4 Hanser created one case for each product with Change and Incident activities.

For the data filtering for finding the Incidents before Change, they removed the cases which does not start with Incident activity, and the cases which does not contain any Change activities. Also they trimmed the cases to contain all activities from the first Interaction events to the last Change event as can be seen on Fig.2.6.

For the data filtering for finding the Incidents after Change, they removed the cases which does not start with Incident activity, the cases which does not contain any Change activities. Also they trimmed the cases to contain all activities from the first Change events to the last Incident event as can be seen on Fig.2.6.

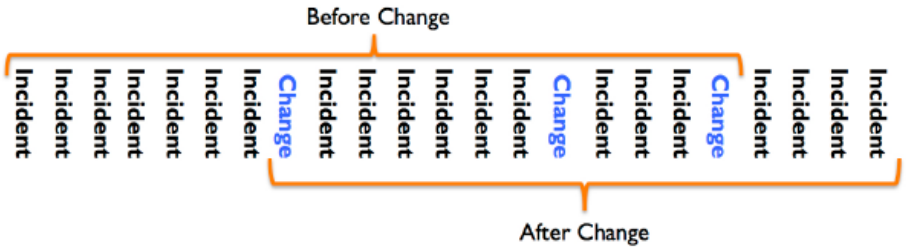


Figure 2.6: Hanser's method to calculate the Incidents Before and After Change.

Chapter 3

Which data model is appropriate for process mining?

In this chapter, we will discuss the criteria that a graph database needs to have to be suitable for process mining analysis. Firstly, we introduce the 2 graph-based data models in Section 3.1. Then we proposed the requirements for a structure should satisfy for Impact Analysis in Section 3.2. Then we compare both data models against these requirements.

3.1 Existing Data Models

Under this section, the summary of Leander's and Esser's data models will be discussed. The general schema, the nodes, the properties will be explained.

3.1.1 Esser's schema

First of all, we will start with Esser's Schema. Fig. 3.1 shows the schema contains one Log node which has a relationship between the Event node. This event node has Directly Follows a relationship with itself to be able to see the proper event relation. Finally, these events should be connected with the entities. Generally, this structure is very easy to interpret.

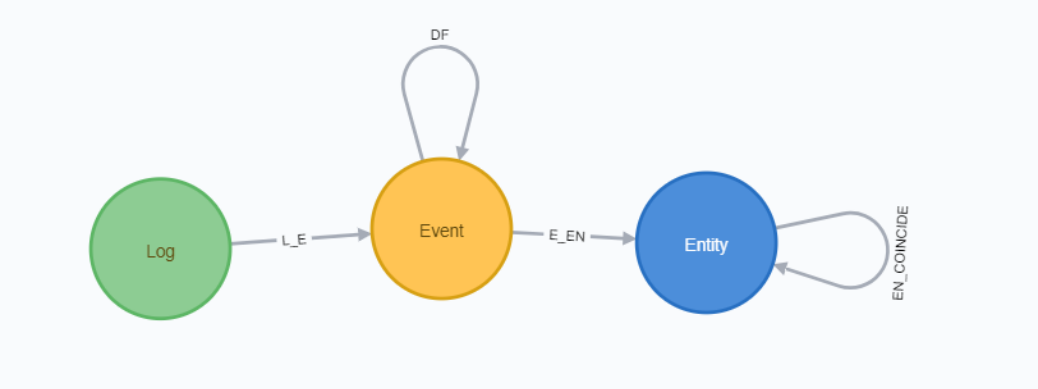


Figure 3.1: Esser's Schema

Fig. 3.6 shows a small example of how Events, Entities and Log nodes are connected. This is an example of CI HMD000053. The blue nodes are Entities, the yellow nodes are Events, and the Log is green. Events and Entities are connected by *E_En* relationship where *L_E* relationship connects Logs and Events. The Event are following each other by *DF* relation.

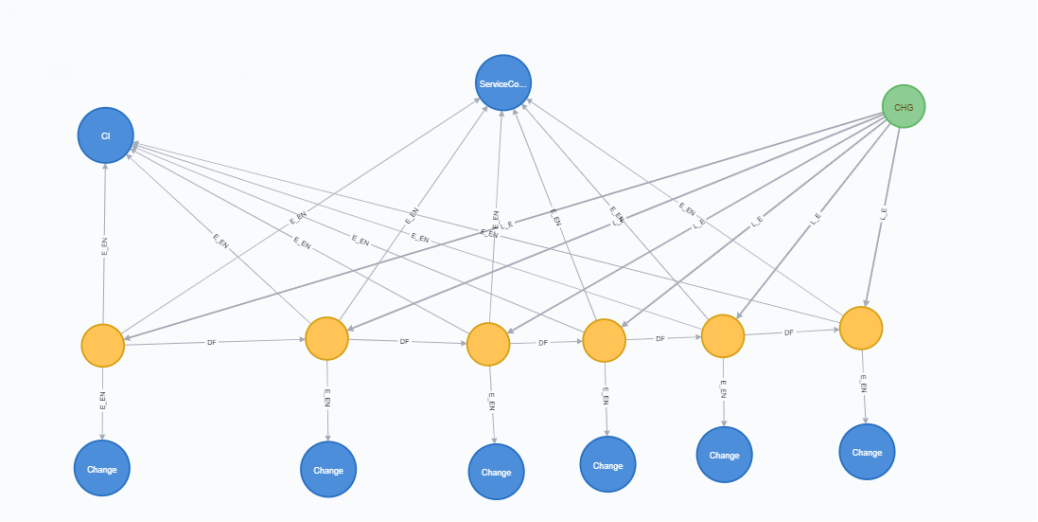


Figure 3.2: The general structure of Esser's data model - CI:HMD000053

3.1.2 Leander 's schema

Fig. 3.3 shows Leander's schema for event data. It contains a Log node that has a relationship with Event node. Event node contains event information that has DF relation with itself and a relation Entity node and to a Common node. Common node connects the same events of two different Entities.

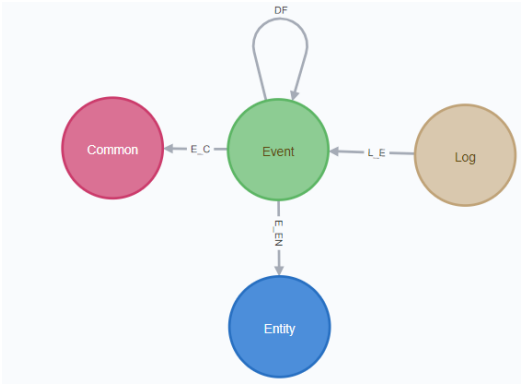


Figure 3.3: Leander's schema

We see same CI example with Esser's Schema, HMD000053. Fig. 3.4 shows a small example of how Events, Entities and Log nodes are connected. This is an example of CI HMD000053. The blue nodes are Entities, the yellow nodes are Events, and the Log node is green. Common nodes are shown in pink. Events and Entities are connected by *E_En* relationship where *L_E* relationship connects Logs and Events. The Event are following each other by *DF* relation. Finally, *E_C* relation links the common and event nodes.

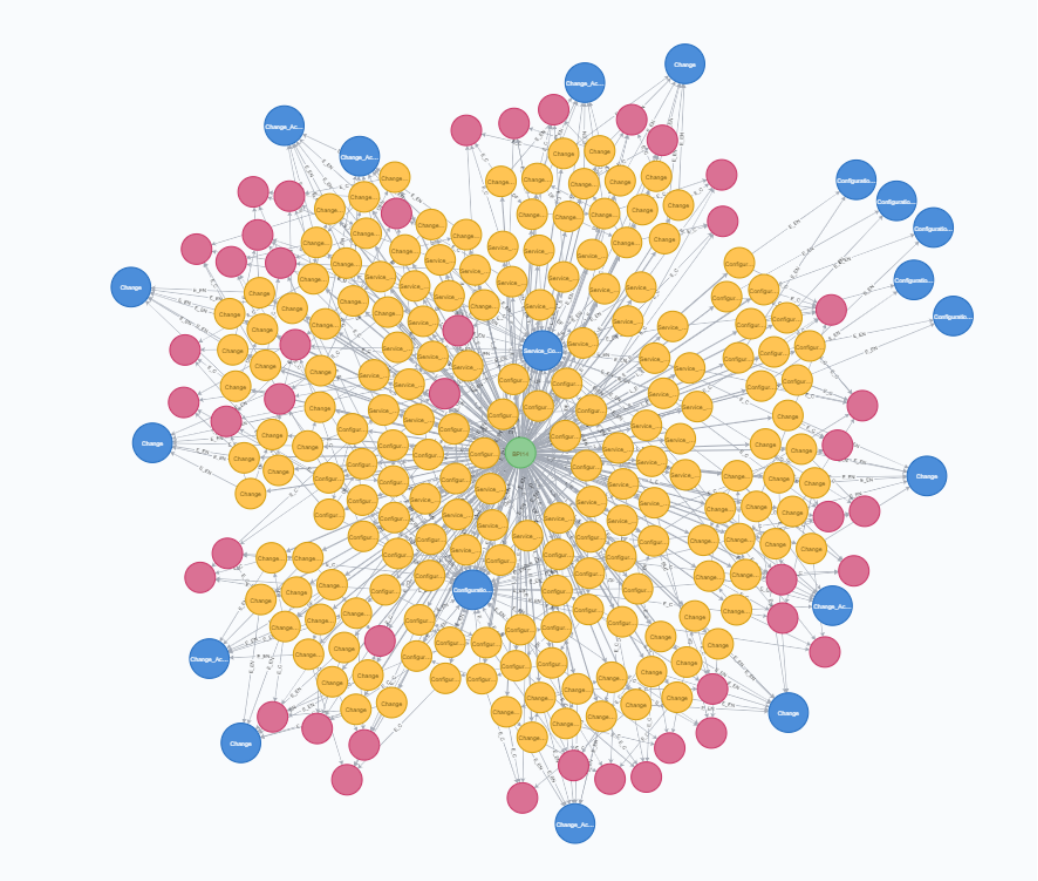


Figure 3.4: The general structure of Leander's data model-CI:HMD000053

In this section, we introduce the data models from Leander and Esser. In the following sections, we will introduce the requirements for a data model should satisfy to conduct the analysis. Then we will give more details about Event and Entity definition.

3.2 Requirements for Impact Analysis on graph-based models

In Section 1.4, we defined our analysis idea. Fig. 1.1 shows the main structure that we would like to achieve for impact analysis. We aim to obtain the events of a Configuration item in order and link the Changes, Interactions, and Incidents to those Events. This will allow us to count how many Interactions/Incidents occur before and after the implementation of a Change. Therefore, this gives us the following requirements:

1. We should be able to query for each CI and receive the Events and Entities in order. Therefore, the data model should propose a complete DF relation between the events of each Entity.

2. We want to receive the DF relation as fast and as easy as possible. Therefore, the queries should be fast and easy to understand.

3. The Entity count should be consistent, there should not any same entities occurring more than one time.

In the Sections 3.3, 3.4 we will give more insights about the Event Definition and Entity Definitions and compare both of the data models. In 3.4 we will also discuss the second requirement. In Section 3.5 presents the comparison between Leander's and Esser's Schema in terms of consistent Directly Follows relation.

3.3 Event Definition

3.3.1 Esser's Schema

Events are the core element of an event log. Therefore, an explanation on events are also needed. Fig. 3.5 shows the event node. The node itself extensively cover many aspects of Event information.

```
{
  "Interaction_idx": 9127,
  "KMNo": "KM0000571",
  "Status": "Closed",
  "Impact": "5",
  "CI_idx": 39562,
  "InteractionID": "SD0009146",
  "RelatedIncident": "IM0003971",
  "Category": "incident",
  "Incident_idx": 4757,
  "Log": "INT",
  "Priority": "5",
  "start": "2013-09-
10T11:15:00+01:00",
  "CINameAff": "LSR000160",
  "FirstCallResolution": "N",
  "CISubTypeAff": "Linux Server",
  "ClosureCode": "Other",
  "ServiceComponentAff":
"WBS000161",
  "end": "2013-09-
10T11:47:00+01:00",
  "Urgency": "5",
  "CITypeAff": "computer",
  "idx": 9127,
  "HandleTime": "156"
```


Figure 3.5: Event definition in Esser's schema

It is crucial to discuss the different attribute types. *Start* and *End* are timestamp attributes. *Log* defines which Log contains this Event.

Since graph databases allow to store the events with multiple case identifiers, it is very essential to discuss how this is presented in Esser's schema. The multiple case identifiers here are *InteractionID*, *CINameAff*, *ServiceComponentAff*, *RelatedIncident*. Fig.3.6 shows how the events are connected to multiple case identifier, which are entities. Yellow nodes are the events and the blue nodes are the entities. The Events and Entities are linked with *E_EN*, and Events are linked by *DF* relation. In Fig.3.6, we see one Configuration Item at the top and Change and Service Components at the bottom.

As can be seen, one event can belong to three different entities.

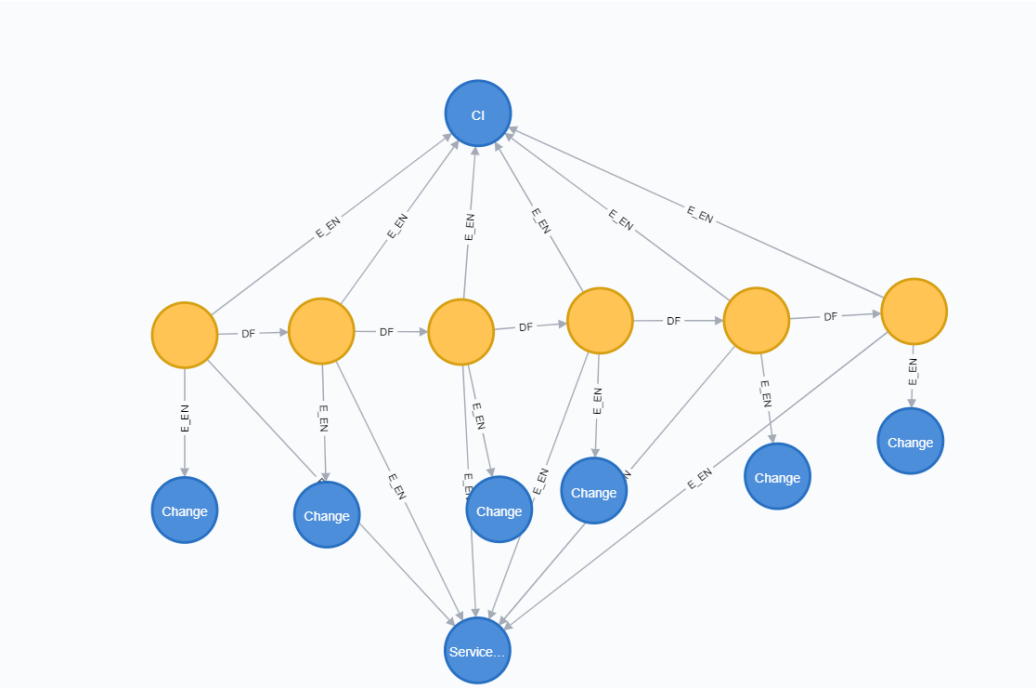


Figure 3.6: The Event nodes are connected with more than one Entity by *E_EN* relation. CI:HMD00053

3.3.2 Leander's Schema

Secondly, it is also crucial to see the Event definition in Leander's schema. In this section, we also introduce Common nodes. Firstly, we explain the event nodes and properties.

Fig. 3.7 shows the event properties that Leander has in the schema. Entity Type and IDraw show which entity type the event belongs to. The *Activity* property shows what is the event executing. *Start* and *End* are the timestamp properties when the activity started and ended respectively.

```
{
  "identity": 1474560,
  "labels": [
    "Event"
  ],
  "properties": {
"EntityType": "Change",
"IDraw": "C00000037",
"Activity": "Change: Scheduled_Downtime_End",
"End": "2013-12-08 05:00:00",
"Start": "2013-12-08 05:00:00"
  }
}
```


Figure 3.7: Event definition in Leander's schema

Each Event are connected to other Events by DF (Directly Follows) relation.

The E_EN (Event to Entity) relationship connects events to entities. Each event is connected to one or more entities and each Entity can have incoming relationships from many events. Leander [10] describes E_En relationship as case identifier of the connected Event, as that Entity was involved in the Event.

The L_ E relationship connects Log nodes to Event nodes. One Log node can belong to one or many Events. This relation indicates that an event is part of the connected log.

Fig.3.8 shows how events are connected with multiple entities. The yellow nodes are events, blue nodes are entities and pink nodes are the common nodes. The picture contains Change, Change_Activity, Configuration Item, and Service Component. The events can be shared by different Entities via Common node. That is how Leander implemented representation of multiple case identifiers on graph databases.

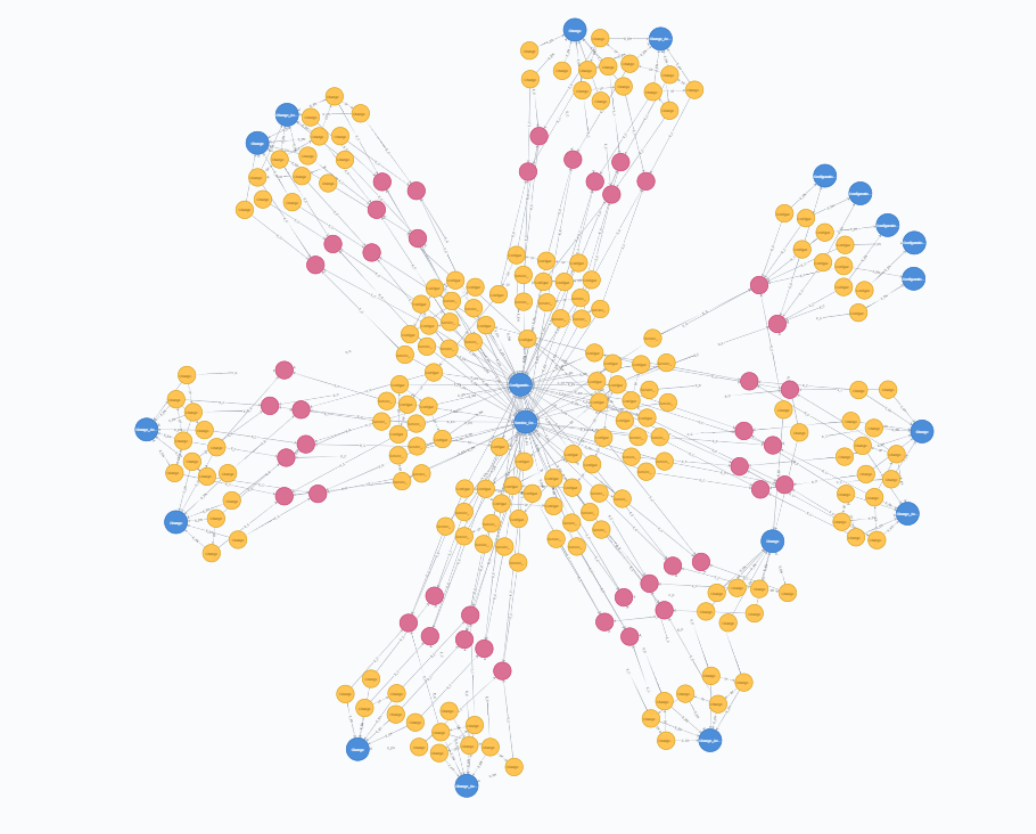


Figure 3.8: Events are connected with common nodes so different entities can share same events. CI:HMD00053

Common node We see that Leander has the approach of presenting events for local entities and Connect the same Events for different entities by Common node. The Common node plays a very important role in terms of making the Events belong to more than one Entities.

When we compare the Event definition between to structures, the Event definition is more clear in Leander's Schema. Such as it very easy to understand what the event is executing by the Activity property and which Entity the event belongs to by EntityType and IDraw property.

On the other hand, Esser provides more information about the Events. It is possible to see which Entities the Events belong to. However, we can not see a clear activity definition of the Events.

3.4 Entity Definition

3.4.1 Esser's Schema

Fig 3.9 shows the properties an Entity has. It has EntityType which defines the entity definition of an entity, IDraw as the ID of the entity. There is Log information which explains from which log the entities are coming from. There are also uID and ID which are concatenated information of Log, EntityType, ID and Log, ID respectively.

Figure 3.9: Entity properties in Esser's Schema

There are five entity types which is possible to obtain with the following query.

```
match (en:Entity) return distinct en.EntityType
```

It is possible to see the entity types in table 3.1

Table 3.1: Entity Types in Esser's schema

Entities
Configuration Item
Change
Incident
Interaction
Service Component

Duplicated Entities The second requirements makes us to check whether the Entities in a path are duplicated. For this comparison our example will be on Configuration Item LSR000160. We concentrate in the Incidents only. To see which Incidents are duplicated Query 1 can be run. Query 1 filters the entities for only Incidents and returning the IDraw of entities and their corresponding counts for all the Incidents.

```
//Query 1 - Find the Duplicated Incident Entities in Schema of
   Esser
//Match the Path for CI: LSR000160
MATCH (n:Entity {EntityType:"CI", IDraw:"LSR000160"}) <
   -[:E_EN]-(ev_inc:Event)--(p:Entity)
// Filter the Entities by Entity Type
where p.EntityType='Incident'
//Return the Entities and the corresponding occurance
return p.IDraw as Entities, count(p.IDraw) as Occurence
```

Fig.3.10 shows that all the duplicated Incident nodes for LSR000160. As can be seen all the Incidents nodes are duplicated.

Entities	Occurence
IM0038376	2
IM0026916	2
IM0026779	2
IM0028659	3
IM0043014	2
IM0020076	3
IM0043759	2
IM0023991	2
IM0025853	2
IM0037116	2
IM0043133	2
IM0035892	2
IM0008813	2
IM0021828	2
IM0003971	2
IM0033906	2
IM0005607	2
IM0005608	2
IM0037280	2
IM0012640	2
IM0006781	2
IM0003846	2
IM0023193	2
IM0037130	2

Figure 3.10: Duplicated Incidents for LSR000160. All of them occur more than 1.

One example of the Duplicated Entities can be seen in Fig.3.11. Green arc shows the Incident IM0026779 and red arc shows the IM0023193.

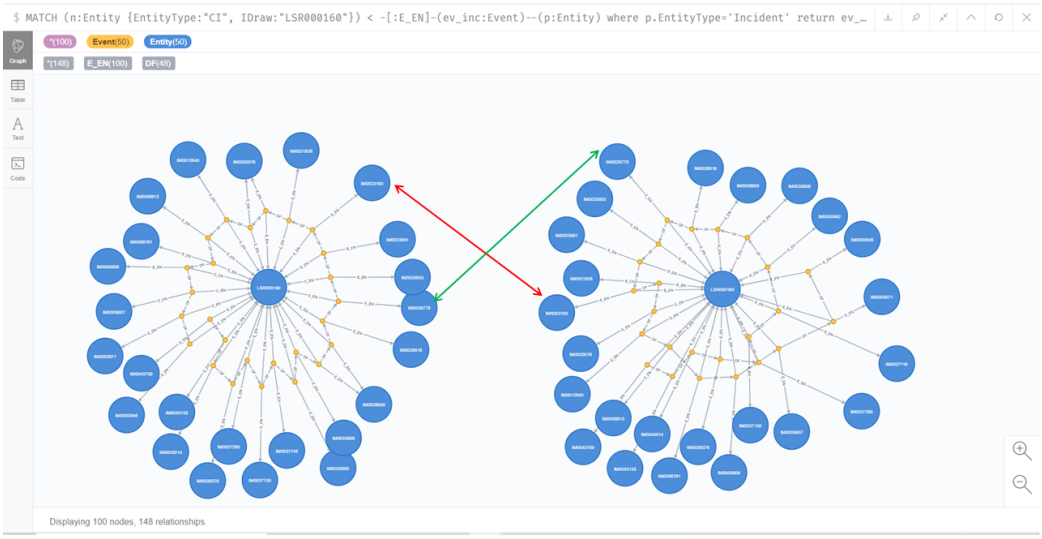


Figure 3.11: Duplicated Incident Information that LSR000160 had.

As we discussed before, each entity should occur once otherwise we can not count properly and it would lead us to analyse wrongly.

3.4.2 Leander's Schema

Fig. 3.12 shows that Entity nodes has nine properties. We will not go through all of them as some of the information is used on the queries. The most important properties for the queries are EntityType and IDraw. These properties are same with the Entity definition in Esser's schema.

```
{
  "EntityType": "Change",
  "uID": "ChangeBPI14C00000003",
  "Emergency_Change": "N",
  "Log": "BPI14",
  "Change_Type": "Release Type 11",
  "IDraw": "C00000003",
  "CAB_approval_needed": "N",
  "ID": "C00000003",
  "IDLog": "BPI14C00000003"
}
```


Figure 3.12: Entity properties in Leander's schema

Table 3.2 shows the all the entity types in Leander's schema. There are nine Entities which is more comprehensive than Esser's schema. However, we will only use Change, Interaction, Incident, and Configuration Item entities in the analysis.

Table 3.2: Entity Types in Leander's schema

Entities
Change Activity
Change
Configuration Item
Incident Activity
Interaction
Assignment Group
Incident
Service Component
Knowledge Document

Duplicated Entities We also check the existence of duplicated Enitites in Jamiro's Schema. The following query should be run for checking the duplicated entities in LSR0000160.

```
//Query 2 - Find the Duplicated Entities
//MATCH the path from CI to the Events of the Entities(In this
   case Incidents)
MATCH (n:Entity{EntityType:'Configuration_Item', ID:'8245'})
--(ev:Event)--(c:Common)
optional match (c) -- (ev_c:Event)
//Filter the Entity Types
where ev_c.EntityType='Incident'
//MATCH the Incidents and the corresponding events
optional match (ev_c) -->(en:Entity)
// Collect the events for each Incident
with en.IDraw as ent_id, collect(ev_c.Activity) as act
where ent_id <> "null"
//Return Each incidents and their occurence
return ent_id as Entities, count(ent_id) as Occurence
```

The result of Query 2 can be seen in the Fig.3.13. The Incidents here occur only once. Therefore, Leander's schema ensures that the Entities are not duplicated.

Entities	Occurence
IM0023991	1
IM0043014	1
IM0037130	1
IM0026779	1
IM0025853	1
IM0043133	1
IM0012640	1
IM0028659	1
IM0006781	1
IM0021828	1
IM0008813	1
IM0005607	1
IM0037116	1
IM0023193	1
IM0003846	1
IM0043759	1
IM0003971	1
IM0026916	1
IM0037280	1
IM0035892	1
IM0038376	1
IM0033906	1
IM0020076	1
IM0005608	1

Figure 3.13: The occurance of each Incidents in 8245- Leander's Schema

The main difference between Esser's and Leander's schema for entities while querying is for the Configuration Items, we have to use the ID property instead of Name. Because one CI can belong to many Service Components at the same time and that results in having the path of a CI in two graph components.

3.5 Directly-Follows Relation

Under this Chapter, we will discuss how the DF relation is defined in two schemas. If we refer to our first requirement we see that there should be a consistent DF relation. We apply two approaches.

1. First, we will clearly state the structure of the queries to receive the

path for a Configuration Item with all the associated Entities.

2. Secondly, we will show how to receive all events of a CI with associated entities of a specific type.

We will show the examples on two different Configuration Items, SBA000015 and LSR000160 in both schemas.

3.5.1 Esser's Schema

In Esser's schema, we will discuss the queries to receive events that happen to a configuration item and the related entities and entities with specific type. The CIs are SBA000015 and LSR000160. The result will be evaluated in terms of receiving correct DF, easiness of the queries and low execution times.

Query 3 gives all the associated Entities for SBA000015.

```
//Query 3 - Receiving Path
MATCH (n:Entity {EntityType:"CI", IDraw:"SBA000015"}) <
   -[:E_EN]-(ev_inc:Event)--(p:Entity)
return ev_inc,p,n
```

Fig. 3.14 explains how the query was structured. Blue nodes are the entities whereas yellow nodes are the events. The starting point should be the Configuration Item(n) and the path should be constructed as starting from the particular CI to the events (ev_inc) and corresponding entities (p).

Fig. 3.14 shows SBA000015 has a complete path. It is experiencing six Changes in order with five changes effecting two different Service Components.

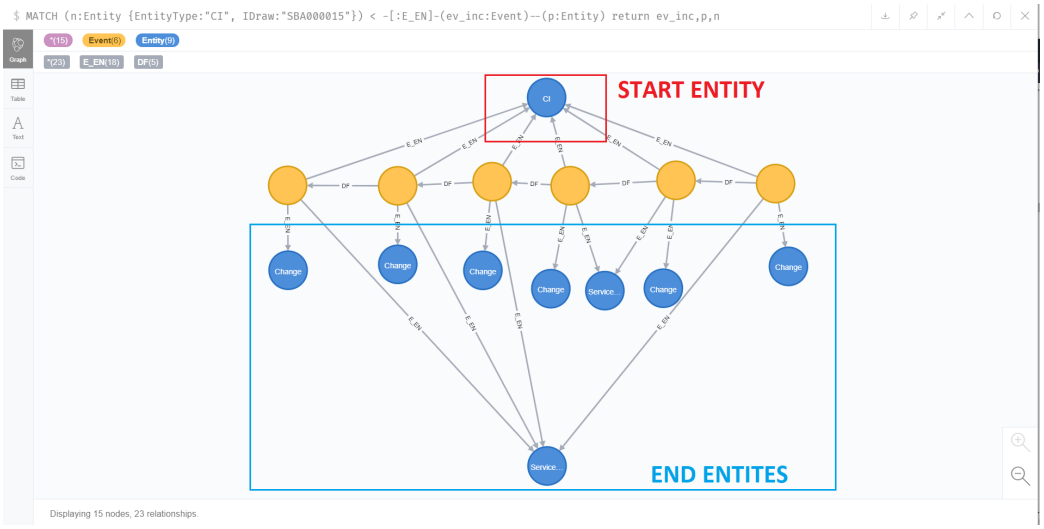


Figure 3.14: Result of Query1 in Esser's Schema-CI:SBA000015

The second approach is receiving all the events that SBA000015 is experiencing but only with related Changes. Therefore, the Service Components will not be obtained anymore.

```
//Query 4 - Receive the Path for Associated Entities
MATCH (n:Entity {EntityType:"CI", IDraw:"SBA000015"}) <
   -[:E_EN]-(ev_inc:Event)--(p:Entity)
where p.EntityType='Change'
return ev_inc,p,n
```

As can be seen from Fig. 3.15 Service Component information is not received anymore, therefore we see the Changes that Configuration Item SBA000015 is experiencing.

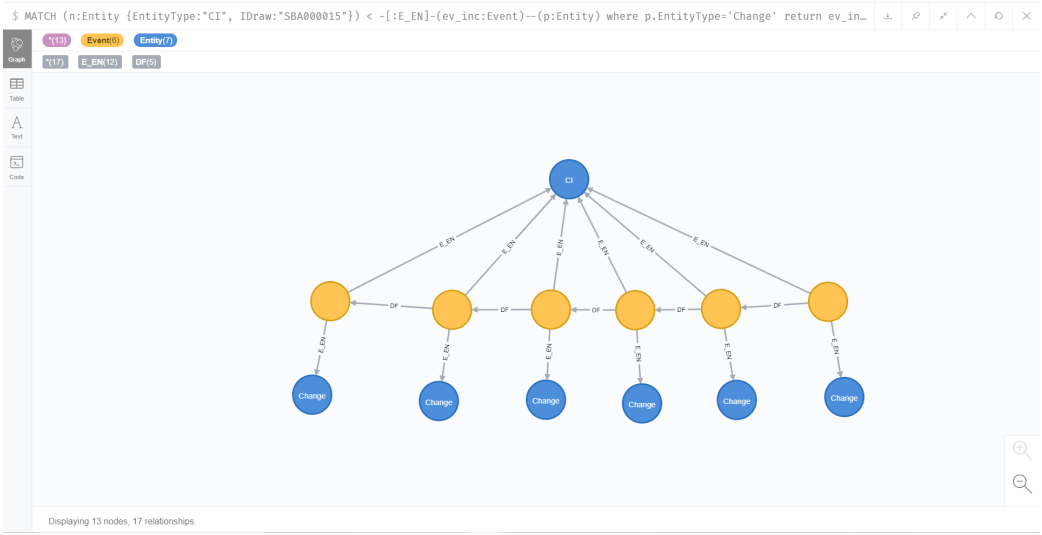


Figure 3.15: Result of Query 4 - Esser's Schema. Only Change entities are returned.

Now, we execute same queries on a more complex CI which has more Events and Entities. We onlyuse the second approach to see whether the simple idea of Query 2 still work on complex cases. Query 5 shows the path for Configuration Item LSR000160. For LSR000160, we will only see Interactions, Incidents and Changes.

```
// Query 5
MATCH (n:Entity {EntityType:"CI", IDraw:"LSR000160"}) <
   -[:E_EN]-(ev_inc:Event)--(p:Entity)
where p.EntityType='Change' or p.EntityType='Interaction' or
   p.EntityType='Incident'
return ev_inc,p,n
```

Fig. 3.16 shows three graph components as result of Query 5. Details are in Appendix F. These components are representing Interactions, Incidents and Changes that LSR000160 is experiencing. In total 83 entities, 56 events and 215 relations were returned.

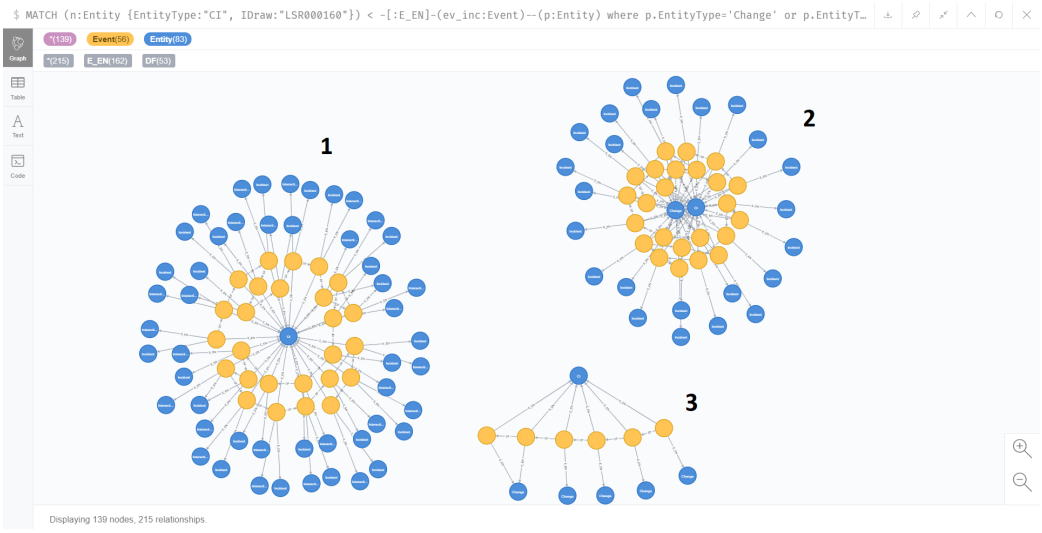


Figure 3.16: Result of Query 5 for Configuration Item LSR000160 with three graph components.

Graph component-1 shows Interactions and Incidents. In total this component has 26 events and 27 entities.

Unlike Graph Component 1, Graph Component 2 only shows the Incidents and one Change with IDraw of 'Unknown'. In total, this component has 26 entities and 25 events.

Graph Component 3 shows the Changes that the Configuration Item is experiencing. Even though there is another Change entity in Graph Component 2 , these are the main Changes LSR000160 is experiencing. This component has 6 events and 6 entities.

As mentioned in Section 1.4, we would like count how many Incidents/Interactions succeed and preceded Changes to conduct analysis. It is very crucial to receive all events happening to a CI. However, instead we get 3 components, hence, this property is clearly violated by the schema. Therefore, we will never get a connected path.

On the other hand, the schema is very self-explanatory and this allows to construct the queries very easily. Moreover, all the queries that mentioned for retrieving the DF relation answers in 2 ms which is very fast.

3.5.2 Leander's Schema

In Leander's schema, we will discuss the queries to receive events that happen to a configuration item and the related entities and entities with specific type. The CIs are SBA000015 and LSR000160. The result will be evaluated

in terms of receiving correct DF, easiness of the queries and low execution times. We will also compare the results with Esser's Schema.

For the first approach, we run Query 6. Query 6 gives the path of Configuration Item SBA00015 under Service Component WBS000161. The ID is 9132.

```
//Query 6 - Receive the Path
MATCH (n:Entity{EntityType:'Configuration_Item', ID:'9132'})
--(ev:Event)--(c:Common)
optional match (c) --(ev_c:Event)
optional match (ev_c:Event) --(en:Entity)
optional match (ev_c)--(en2:Entity)
return n,ev,c,ev_c,en,en2
```

Query 6 results as 6 Entities and 56 Events. There is only one graph component as can be seen in Fig. 3.17. Event nodes are yellow, blue nodes are entities and pink nodes are common nodes. As can be seen, Every Entity has its own local Events.

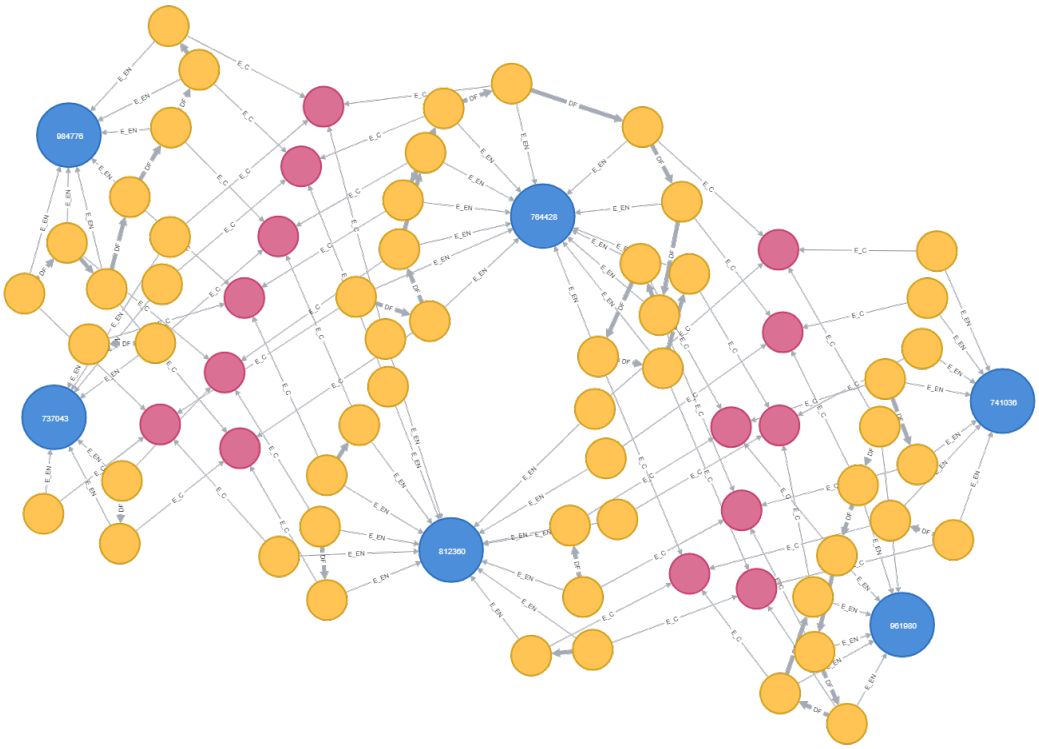


Figure 3.17: Result of Query 6 - Path of 9132. SBA000015

We will not apply the second approach as the Configuration Item is very

simple, and filtering the Entities would not add value.

Another Configuration Item for the sake of comparison is LSR000160 with ID of 8245. We run Query 7 for receiving the path for LSR000160. Firstly, it results in a very large picture as can be seen in Fig. 3.18. It returns 1550 nodes.

```
//Query 7
MATCH (n:Entity{EntityType:'Configuration_Item', ID:'8245'})
--(ev:Event)--(c:Common)
optional match (c) --(ev_c:Event)
optional match (ev_c:Event) --(en:Entity)
optional match (ev_c)--(en2:Entity)
return n,ev,c,ev_c,en,en2
```

Secondly, it runs in excessive times. The reason it looks this complicated and takes too much time is because it shows the Entities; Configuration Item, Service Component, Knowledge Document, Interaction, Incident, and Change. However, since most exciting parts for us are only Incident, Interaction, and Change, it is necessary to filter the events by Event Type as it is in Query 8.

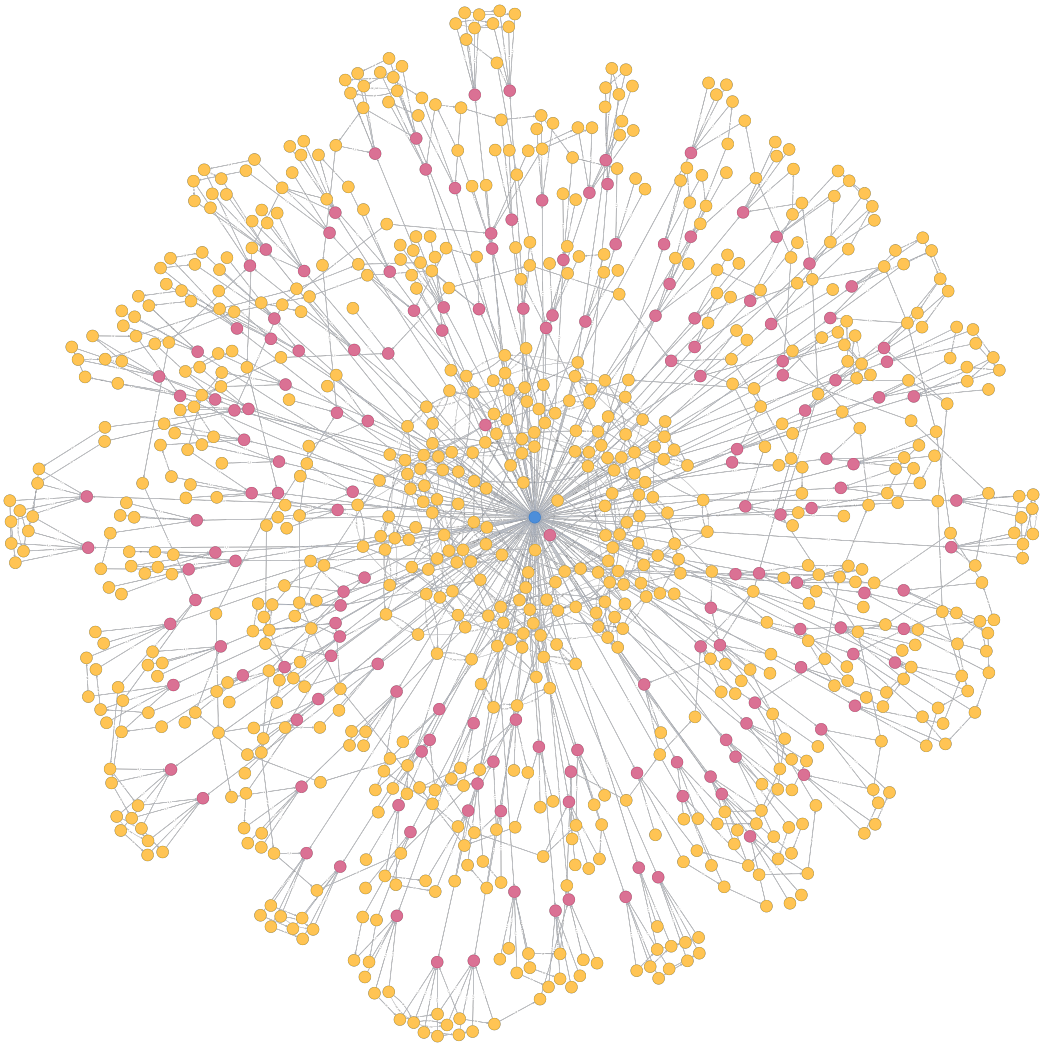


Figure 3.18: Path of 8245.

```
//Query 8 - Receive the Path for associated Entities
//Match the related path of Configuration Item
MATCH (n:Entity{EntityType:'Configuration_Item', ID:'8245'})
   --(ev:Event)--(c:Common)
optional match (c)--(ev_c:Event)
//Take only the events related to Change, Interaction and
   Incidents.
where ev_c.EntityType='Change' or
   ev_c.EntityType='Interaction' or ev_c.EntityType='Incident'
//Match the DF relation between ev_c and followed Events
optional match (ev_c)- [r:DF]->(ot:Event)
// Take only the events related to Change, Interaction and
   Incidents.
where ot.EntityType='Change' or ot.EntityType='Interaction' or
   ot.EntityType='Incident'
//Match the entities
optional match (ev_c)--(en:Entity)
// Take only Change, Interaction, and Incident.
where en.EntityType='Change' or en.EntityType='Interaction' or
   en.EntityType='Incident'
return ev,c,ev_c,en,r
```

The result of Query 8 can be seen in Fig. 3.19. The result contains 550 nodes and is received in only four seconds.

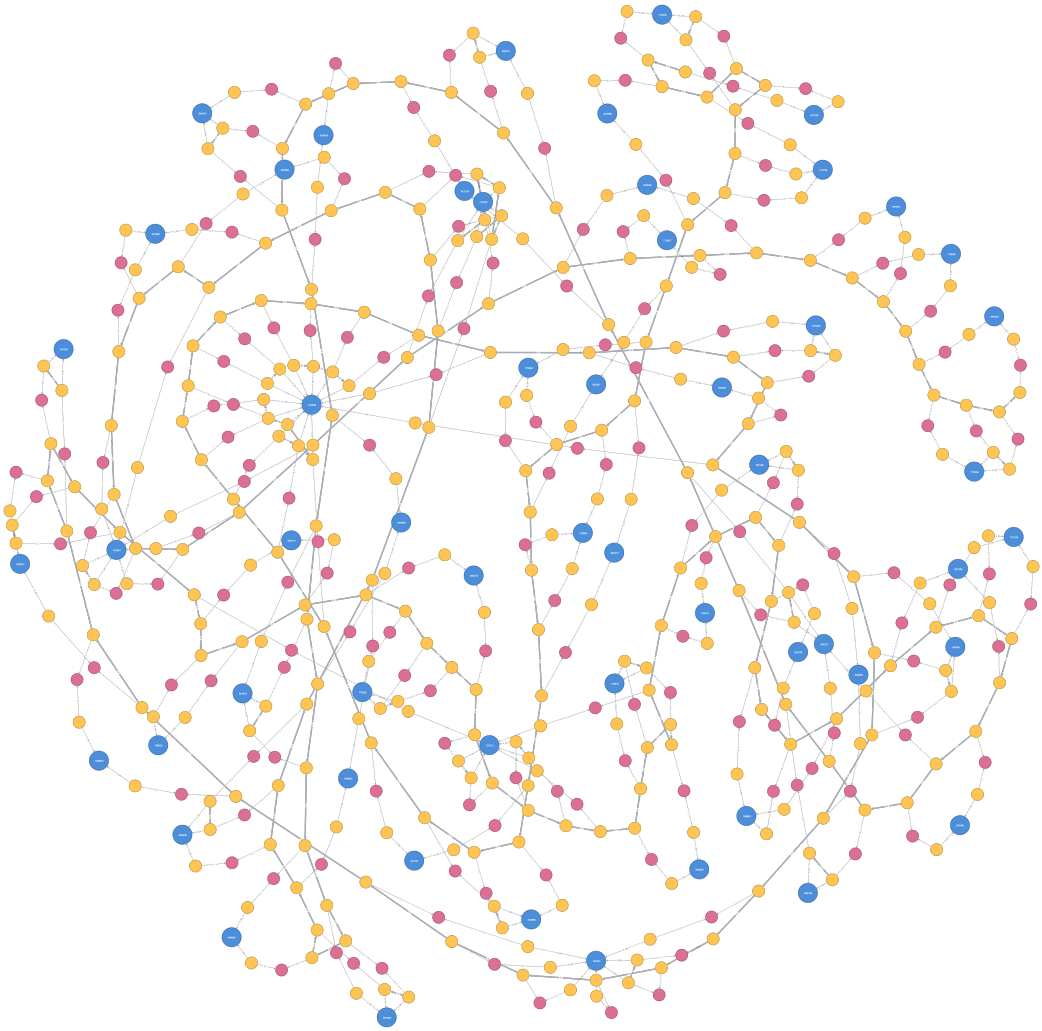


Figure 3.19: Result of Query 7 - Path of CI 8245 with only Change, Interaction, and Incident.

The main challenge for this schema is to understand the common nodes as it is very unusual but very useful for connecting the same Events for different Entities with avoiding any information loss and any duplication. To conclude, Leander's schema offers a complete path and correct Directly Follows relation between entities. However, filtering is necessary for avoiding excessive execution times.

3.6 Conclusion

The simplicity and correctness for the data model and simplicity and fastness of the queries were compared. Leander's schema had a complete DF relations which was missing in Esser's schema. However, it was simpler to construct and faster to run the queries in Esser's schema whereas intense filtering was needed in order to run the queries faster in Leander's schema. Moreover, Leander's schema provides more accurate event definition than Esser's schema does. Therefore, for a process mining analysis Leander's schema is more suitable.

Chapter 4

Which queries and analysis steps are necessary to execute an impact analysis on graph data?

In Chapter 3, we select Leander's data Schema for the analysis. In this chapter we first discuss the business understanding which we revisit the concept of impact analysis. Then, we give data understanding and preparation for the analysis. Then we give the Analysis Idea and present our Analysis Schema.

4.1 Business Understanding

In Chapter 2, we present BPIC14 questions and we mentioned about our analysis. We conduct an impact analysis where we calculate the number of Incidents/Interactions before and after a Change.

As can be seen on Fig.1.1, we count the number of Incidents and Interactions between Changes. Therefore, we count the Entities not the Events.

However, Leander's Schema orders Events over time, not the Entities. Hence, we propose a solution to order the Entities of a related Configuration Item depending on the timestamp.

4.2 Data Understanding

In Chapter 3, we explained the structure of the data. Events are on a DF-path per Entity. Common nodes describe Events and each Event node locally

linked to one Entity. When Events are linked to the Common nodes that means Events are involved in multiple entities Now, we give more insights about the data. The data that we will work on is between September 2013 and March 2014. In total it covers six month long information. The number of Configuration Items, Changes, Interactions, and Incidents can be seen in Table 4.1.

Table 4.1: The number of Configuration Item, Change, Incident, and Interaction.

Entities	Total Count
Change	18000
Configuration Item	15063
Interaction	147006
Incident	47004

4.2.1 Exporting Events into sequential Event Log

Before the analysis, we should observe any problems related to the data an propose and effective solution. To obtain these insights, we use the existing process mining tool ProM. First we export the data into an Event log format that can be read by ProM. We export each Entity node as a record with the attributes of Configuration item ID, Entity ID, Last timestamp of the Entities. We exported this into a CSV file.

To be able to create a log file for each CI, Query 9 should be run. Query 9 orders each Entity that a Configuration Item is experiencing then store these in CSV file called *log_graph.csv* by using *apoc.export.csv.query* command.

```
// Query 9 - Export the Entities in order to CSV file.
CALL apoc.export.csv.query("MATCH
   (n:Entity{EntityType:'Configuration_Item'})
   --(ev:Event)--(c:Common)
optional match (c)--(ev_c:Event)--(en:Entity)
where en.EntityType='Change' or en.EntityType='Interaction' or
   en.EntityType='Incident'
with en.EntityType as ent, en.IDraw as ID, ev.End as date,
   n.ID as CI order by date
with ent, ID, min(date) as min_date, max(date) as max_date, CI
return CI,ID,ent,min_date,max_date
order by max_date","log_graph.csv",{stream:true})
```

To understand this query better, we can analyze Query 9 line by line. The following part of the query takes all the Configuration items and filtering the Entities by EntityType. This filter here is not necessary, as this is a log file and can include every information, but since we are mainly interested in Changes, Interactions, and Incidents, we used a filter.

```
MATCH (n:Entity{EntityType:'Configuration_Item'})
   --(ev:Event)--(c:Common)
optional match (c)--(ev_c:Event)--(en:Entity)
where en.EntityType='Change' or en.EntityType='Interaction' or
   en.EntityType='Incident'
```

Then WITH clause is used to take a subset of the properties, in this case EntityType, IDraw of the Entities, End date of the Events, and CI ID, and to forward the necessary properties to ORDER BY clause. ORDER BY orders each Entity based on the completed time of that entity per Configuration Item.

```
with en.EntityType as ent, en.IDraw as ID, ev.End as date,
   n.Name as CI
order by date
```

After ordering the entities per Configuration Item, the second WITH clause is necessary to order the entities based on the latest date of all of them.

```
with ent, ID, min(date) as min_date, max(date) as max_date, CI
return CI,ID,ent,min_date,max_date
order by max_date
```

The result is exported to a CSV file called "log_graph.csv" and it can be uploaded to ProM 6.9. Details are in see Appendix A.

4.2.2 Visualize Exported Event Data in ProM

We can analyze the Dotted Chart to gain insights about the data. For details about using ProM, please refer to Appendix A.

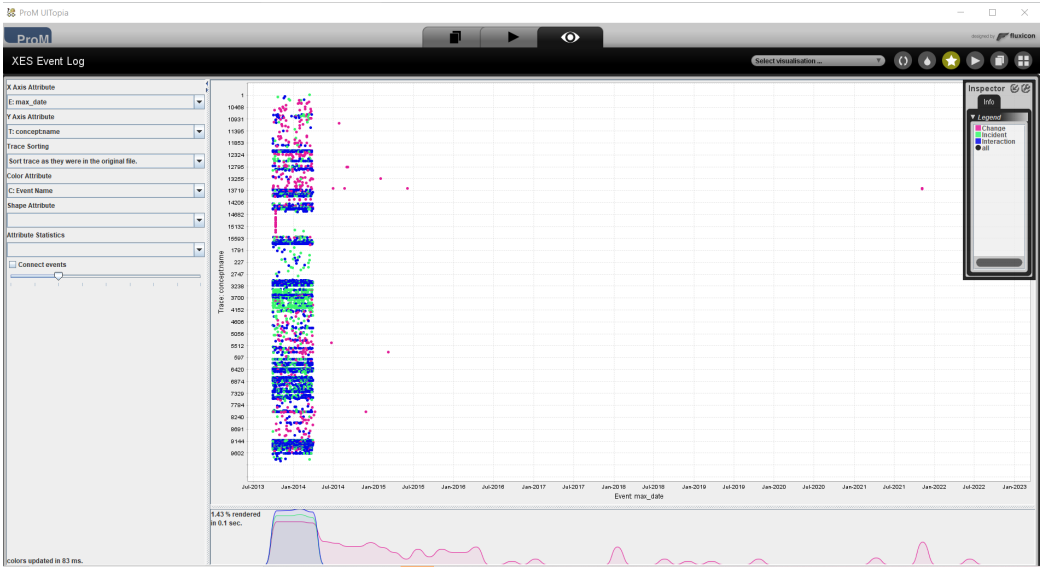


Figure 4.1: Caption

As can be seen on Fig.4.1, there are Entities which are ending after March 2014 which is the ending date of the data. When these Entities were cross checked with the data, it was found that these specific activities are *Change: Planned Start*, *Change: Planned End*, *Change: Requested End*. These activities are assumed that they are system generated. Therefore, there is no need to include them in the analysis.

Furthermore, when we checked the raw data, we saw that there are duplicated records which also reveals itself in the graph schema. These events would not have a big impact in the analysis but in terms of data quality these events should not be there.

4.2.3 Data Preparation

First of all, the data processing is different than Hanser's method. The details are in 6.5.

We identified that there are Events which are happening after the end date of the data. When we checked, we found that they are system generated nodes and there is no need to keep them a they do not add value to our analysis. We also found that there are duplicated event nodes. However, in the graph we have the DF-relation between the Events, therefore, we can not just remove the Event nodes.

There are 2 options:

1. Create a new DF_Mat relation

2. Delete the Events and recompute the DF relation

Materialize new DF Relation

To remove the system generated Events from the path of Events, one solution is defining a new relation called DF_Mat between all the events except *Change: Planned Start*, *Change: Planned End*, *Change: Requested End*. Fig. 4.2 shows the DF-Mat relation between the events except *Change: Planned Start*, *Change: Planned End*, *Change: Requested End*.

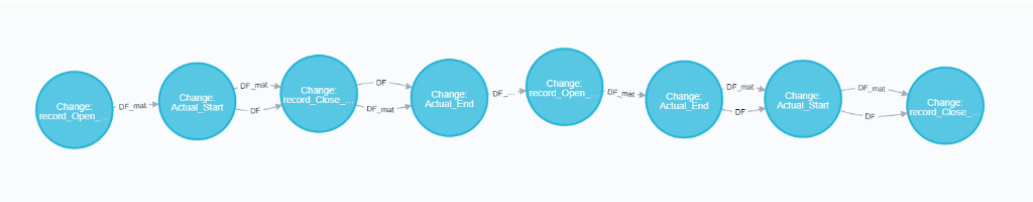


Figure 4.2: DF-Mat relation between the events.

To be able to to that, Query 10 should be run for each Configuration Item in the database. Query 10 creates the DF_Mat relationship by eliminating events *Change: Planned Start*, *Change: Planned End*, *Change: Requested End* and creates relationship in order of start date of the events. End date also could be used, as both are holding same information. This query should run in a python script for every Configuration Item.

Before running the query, it is important to assign an ID to Event nodes to be able to select the unique nodes. We use **id(node)** property and assign the id's to every event in the database. The *dbms.memory.heap.max_size* should be increased to 20 GB otherwise the query can take too much time. As a result, 4926161 properties are created.

```
MATCH (e:Event) SET e.id = id(e)
```

```
\\Query 10 - Create a relationship between the desired nodes.
MATCH (n:Entity{EntityType:'Configuration_Item', ID:'%s'})
   --(ev:Event)--(c:Common)
optional match (c)--(ev_c:Event)--(en:Entity)
where en.EntityType='Change'
with n.ID as CI_ID, n, en.EntityType as ent,ev.Activity as
   act, ev_c,en.IDraw as ID, ev.Start as date, ev
where not act="Change: Planned_Start"
with ev_c, ent, act, ID, date, CI_ID, n, ev
where not act="Change: Planned_End"
with ev_c, ent, act, ID, date, CI_ID, n, ev
where not act="Change: Requested_End_Date"
with CI_ID, ent, act, ev.id as nodeID, ID, date
order by date
with collect(nodeID) as nodeList
unwind range (0, size(nodeList)-2) as i
with nodeList[i] as first, nodeList[i+1] as second
match (ev1:Event), (ev2:Event)
where ev1.id=first and ev2.id=second
create (ev1)-[:DF_mat]->(ev2)
```

After matching the path with the properties needed, it is very essential to use "Where not" command to filter the undesired events. Then the rest of the events should be ordered by the date. The most important part of this query with the comments is as following:

```
//The wanted events should be collected into a list
with collect(nodeID) as nodeList
//Every element of the list should be taken individually.
unwind range (0, size(nodeList)-2) as i
//Every consecutive events should be taken
with nodeList[i] as first, nodeList[i+1] as second
//The events where the ids are equal, should be matched
match (ev1:Event), (ev2:Event)
where ev1.id=first and ev2.id=second
//The relationship can be created.
create (ev1)-[:DF_mat]->(ev2)
```

Creating a new relationship can be hard to work with as it is not offering a convenient filtering on the events. If such events are preferred to obtain, then

an intense filtering would be needed. Therefore, for both of the problems, we can offer another solution which could be more easier to handle.

Delete and Recompute Df Relation

With deleting the problematic and duplicated events and the relation between them, we could solve both of the problems, duplicated events and system generated events, in the data. The steps are as follows:

1. Delete the DF relation
2. Delete the duplicated and system generated nodes.
3. Create the DF relation again.

Deleting DF relation We have to delete the DF relation otherwise it would be impossible to delete the nodes as they would hold relation.

Since the following query collecting every record in the disk first, it is essential to increase the dbms.memory.heap.max_size to 20 GB. Then 4199682 relationship would be deleted.

```
match (ev:Event)-[r:DF]->(ev2:Event) delete r
```

Deleting duplicated and system generated nodes After deleting the relation, it becomes possible to delete all the unwanted nodes. To delete the duplicated nodes, the following query should be run. It is collecting all the duplicated nodes and deletes the 'TAIL' of the list. And there is only one node left.

```
// Query 11 - Delete the Duplicated Events
MATCH (en:Entity{EntityType:'Configuration_Item'} )--
   (n:Event{EntityType:'Configuration_Item'})--(c:Common)
--(ev:Event)
with ev.IDraw as ev, n.IDraw as or_ev_idraw, n.Activity as
   act, collect(n) as col
where size(col)>1
FOREACH (n in tail(col) | DETACH DELETE n)
```

To understand better, we can detail the query more. The match command matches the events for each CI and entity and collect the same activities for each event as list.

```
MATCH (en:Entity{EntityType:'Configuration_Item'} )--
   (n:Event{EntityType:'Configuration_Item'})--(c:Common)
--(ev:Event{EntityType:'Change'})
with ev.IDraw as ev, n.IDraw as or_ev_idraw, n.Activity as
   act, collect(n) as col
```

The last part checks whether the list contains more than one event. If so, it deletes all of the events and leaves only one.

```
where size(col)>1
FOREACH (n in tail(col) | DETACH DELETE n)
```

Finally, 1047 nodes and 3141 relationship was deleted. To delete the system generated nodes, the following query should be executed. It deletes 90k nodes.

```
//Deletes system generated nodes for changes.
MATCH (ev:Event{EntityType:'Change'})
where ev.Activity='Change: Planned_Start' or
   ev.Activity="Change: Planned_End" or ev.Activity='Change:
   Requested_End_Date'
detach delete ev
```

Creating the DF relation again After deleting erroneously recorded and duplicated nodes, we recompute DF from timestamps and correlated information. For that, we will run Leander's query which was used to create the DF relation at the beginning.

```
MATCH (n:Entity)
MATCH (n)-[]-(ev)
WITH n, ev as nodes ORDER BY ev.Start, ID(ev)
WITH n, collect(nodes) as nodeList
WITH n, apoc.coll.pairsMin(nodeList) as pairs
UNWIND pairs as pair
WITH n, pair[0] as first, pair[1] as second
CREATE (first)-[df:DF]->(second)
SET df.EntityType = n.EntityType
SET df.EntityId = n.ID
RETURN null
```

We proposed two options for working with the erroneously recorded events. First one is executing a projection operation and computing DF_Mat between the desired events. Second one removes the undesired nodes from the database. We choose to execute the second option because that would also solve the excluding duplicated nodes in the database. If it is desired to keep the information in the database, then creating a new relation would be necessary. However, if there is no need store the information, deleting operation would be more easier and efficient to construct.

4.3 The Analysis Idea

After processing the data and remove the defectively recorded events, the data is ready to conduct the analysis.We propose a new schema called Analysis Schema to run the impact analysis. If we revisit the impact analysis provided in BPIC14, we see that the explanation of the question defines the identification of Impact Pattern as the number of **Closed** Incidents/Interactions that a Change causes. We need to order the Entities by the Last Timestamp. We create a new Analysis Schema. Then we calculate the number of Incidents and Interactions before and after the changes on Analysis Schema. Also, as described on Section 2.2.3, Hanser created one case for each CI with Change and Incident/Interaction activities. After creating the Analysis Schema, each CI will become a case.

The following figure explains the real representation of creating Analysis Schema. There is one CI and 5 different entities. The LT stamp on the Events represents the Last timestamp of the particular Entity. They are ordered as LT1, LT2, LT3, LT4, LT5. Therefore, ENT1 is the first Entity in the path, ENT5 is last the last.

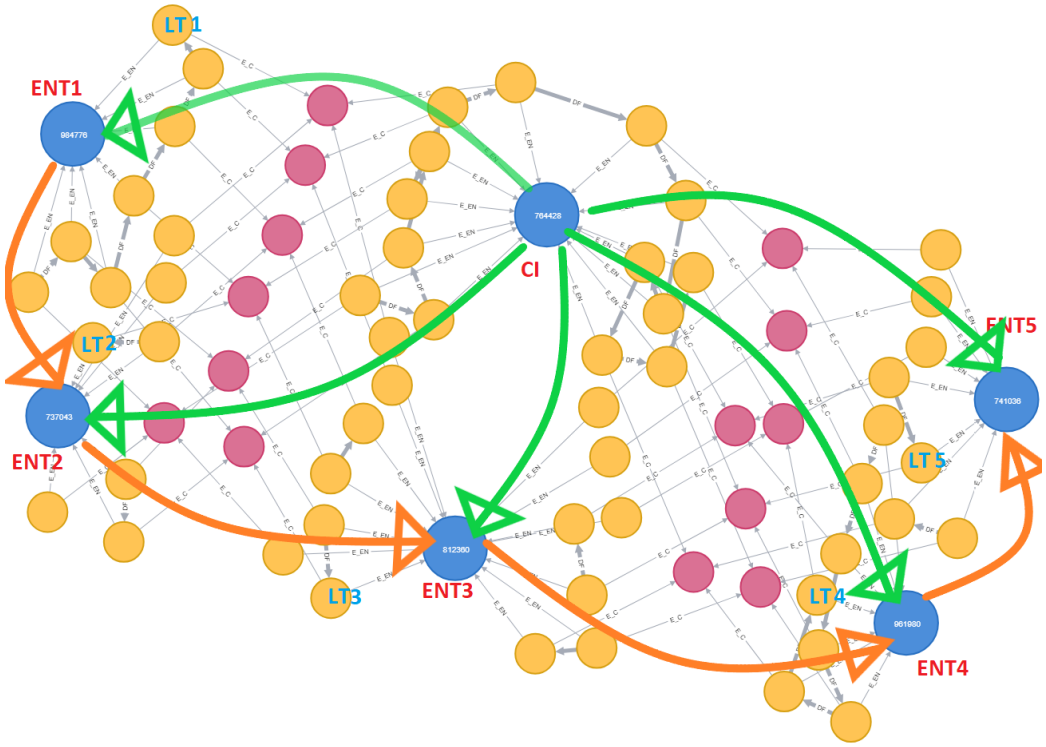


Figure 4.3: The real path of 9132 with one CI and 5 Entities. The orange arc is representing *:has* relation between CI and Entities, green arc represents *en_DF* relation between Entities.

The outcome from 4.3 is the Analysis Schema which is shown in Fig.4.4.The details are discussed in section 4.4, 4.5, 4.6.

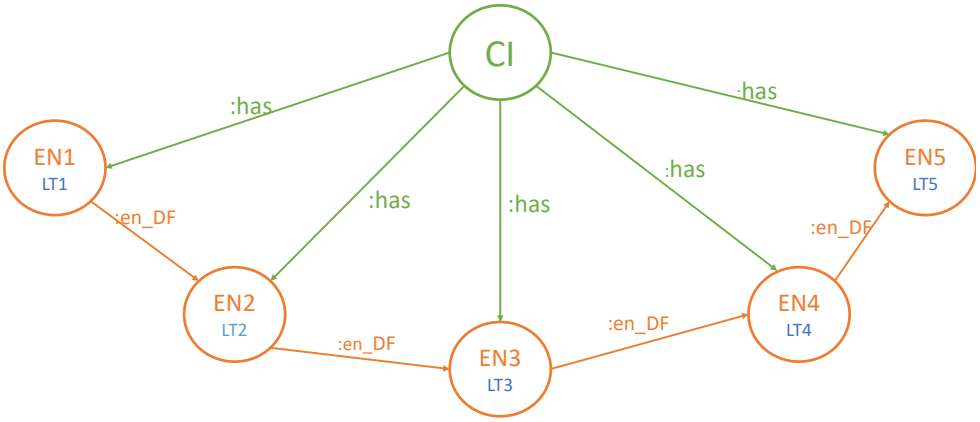


Figure 4.4: The CI *has* its own entities in order which are connected with *en_DF* relation.

4.4 Aggregating all events per entities

To create the Analysis Schema,it is required to aggregate events in one node.

Before giving the exact query, we can discuss the overview of the query. As can be seen in Fig.4.5, our start entity will be the Configuration Item, end entity will be the entities the Configuration Item is experiencing. While aggregating the Events, we will work on the Events in the blue box.

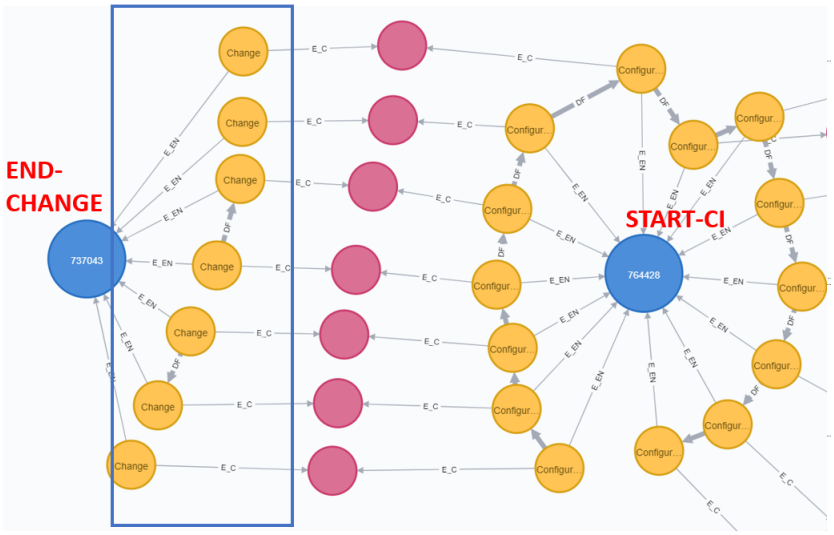


Figure 4.5: Start node is Configuration Item, end node is Change. We will work on the events in the blue box.

It is also important to discuss how to order the aggregated events; based on First Timestamp or last Timestamp. Before diving into the discussion, it is essential to give the background for First and Last Timestamp. Fig.4.6 shows the First and Last timestamp of the Events.

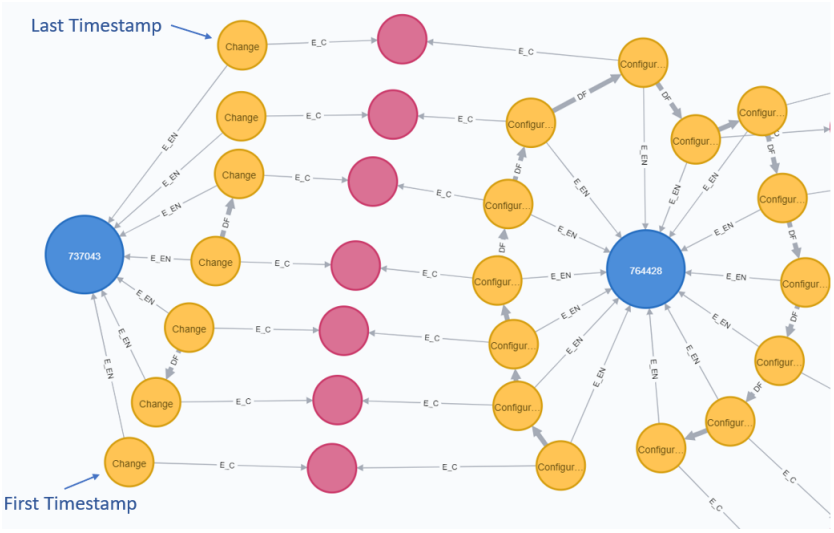


Figure 4.6: First and Last timestamps of the aggregated events.

The identification of Impact Pattern is defined as the number of **Closed** Incidents/Interactions that a Change causes. This is our first motivation why we order the aggregated events based on their last timestamp.

Second motivation is lack of information of why an Incident or Interaction is closed such as because of the completed Change or anything else. To elaborate more, we think present an example scenario where an Incident starts before a Change and closes after the next Change starts as can be seen in Fig.4.7.

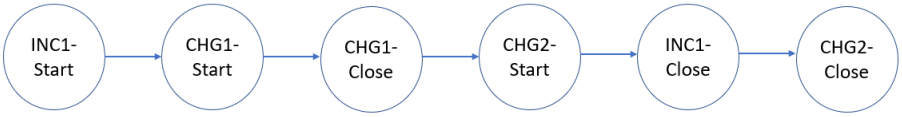


Figure 4.7: Inc1(Incident 1) is closed after CHG1(Change 1) is closed and CHG2(Change 2) is started

Then the ordered aggregated events could be as following if we order them based on the First timestamp.

Figure 4.8: The order based on the First Timestamp.

Incident 1 is closed after Change 1 but in the order that information will not take place, instead Change 1 will be interpreted as it did not have any effect on Incident 1.

On the other hand, if the previous scenario taken for ordering the based on Last timestamp, then the order would be as follows:

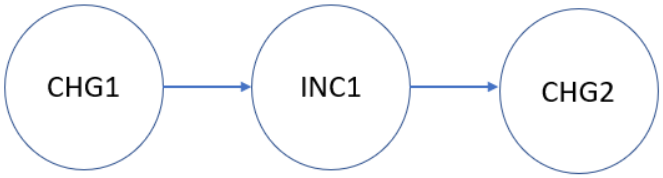


Figure 4.9: The order based on Last Timestamp.

The first scenario might be true, but we do not have the related information. Hence, we order the aggregated events depending on the Last Timestamp. However, we give the queries for both options.

Materialize Relations Into new Entity Records

Under this section we query for the records which we use later to define new Entity nodes in Analysis Schema. The desired records that wanted to keep in the new Entity nodes in Analysis Schema as follows:

- Entity Type
- Configuration Item ID
- Entity ID
- Originated From (Only for Changes)
- Completion Time

The Entity ID and Configuration Item ID is essentially a primary-foreign key relationship from the Configuration Item to the Entity with a time stamp. This will allow to materialize this primary-foreign key relation into a new Entity node.

Between the Entities we create en_DF relation. There are two options which were discussed in 4.4. We can order Entities either based on First Timestamp or Last Timestamp. We provide the queries for both of the options. To be able to create the en_DF based on First Timestamp, Query 12 should be run.

```
//Query 12 - Order the Entities in First Timestamp
MATCH (n:Entity{EntityType:'Configuration_Item', ID:'%s'})
   --(ev:Event)--(c:Common) optional match
   (c)--(ev_c:Event)--(en:Entity)
where en.EntityType='Change' or en.EntityType='Interaction' or
   en.EntityType='Incident'
with en.EntityType as ent,ev_c.Activity as act, en.IDraw as
   en_ID, ev_c.Start as date, n.Name as CI_name,
   n.Service_Component as SC, n.Type as CI_type, n.ID as CI_ID
order by date
with en_ID, date, act, ent, CI_name, SC, CI_type, CI_ID
where not ent='null'
return ent,en_ID, CI_name, CI_ID, SC, CI_type, min(date)order
   by min(date)
```

Firstly the events between shared Configuration Item should be matched with the MATCH command. Then the entities should be filtered for either Interaction or Incident. Then, WITH clause should take place to take the events based on the EntityType and IDraw. The events should be ordered by start date in case for two events are happening at the same time. WITH the ID of the entities, events, start date, and EntityType, a filtering should take place to filter out the null type entities because of the Entities filtered at the first place. Then collected Events for the same Entities should be returned with their corresponding entity name with ordering based on the date.

To be able to create the en_DF based on Last Timestamp, Query 13 should be run.

```
//Query 13 - Order the Entities based on Last Timestamp
MATCH (n:Entity{EntityType:'Configuration_Item', ID:'%s'})
   --(ev:Event)--(c:Common) optional match
   (c)--(ev_c:Event)--(en:Entity)
where en.EntityType='Change' or en.EntityType='Interaction' or
   en.EntityType='Incident'
with en.EntityType as ent,ev_c.Activity as act, en.IDraw as
   en_ID, ev_c.End as date, n.Name as CI_name,
   n.Service_Component as SC, n.Type as CI_type, n.ID as CI_ID
order by date
with en_ID, date, act, ent, CI_name, SC, CI_type, CI_ID
where not ent='null'
return ent,en_ID, CI_name, CI_ID, SC, CI_type, max(date)
order by max(date)
```

Query 13 is executed for each CI. The parameter %s shows different Configuration Items as a Python has been used to run this query for each CI from a CSV file all CIs are stored.

The result of Query 13 can be saved in CSV file for different CIs to process it later to create the graph. Fig.4.10 shows the result of Query 13 on CI 5613. The Changes, Inetractions and Incidents are in order.

ent	en_ID	CI_name	CI_ID	SC	CI_type	max(date)
Change	C00002217	SBA000716	5613	WBS000105	application	10/4/2013 12:00
Change	C00002407	SBA000716	5613	WBS000105	application	10/7/2013 18:00
Change	C00004088	SBA000716	5613	WBS000105	application	10/23/2013 15:00
Change	C00005437	SBA000716	5613	WBS000105	application	11/6/2013 10:30
Change	C00007175	SBA000716	5613	WBS000105	application	11/22/2013 15:00
Change	C00007590	SBA000716	5613	WBS000105	application	11/28/2013 9:30
Change	C00007748	SBA000716	5613	WBS000105	application	12/16/2013 9:13
Change	C00006339	SBA000716	5613	WBS000105	application	1/2/2014 8:07
Change	C00010017	SBA000716	5613	WBS000105	application	1/6/2014 18:00
Interaction	SD0140382	SBA000716	5613	WBS000105	application	3/21/2014 16:51
Incident	IM0045382	SBA000716	5613	WBS000105	application	3/21/2014 16:51

Figure 4.10: Result of Query 13 - CI: 5613

Moreover, we also export the *Originated from* record from Entity Change_Activity as in the Analysis we use this information to understand the reasons for each Change. This property will be stored in a CSV file for every Change. We will upload this records separately.

DF relation to Entity Records

To be able to upload to a graph, there should be another column in the CSV files for each CI which indicates the DF relationship between the aggregated events. The entity records returned by query 13 are sorted by time. For each entity record, the next Entity record is also the next following one. Therefore, we create another column called "en_DF","en_name" and "en_DF_Date" respectively as can be seen in Fig.4.11. Under "en_DF" we keep the ID of new Entity, en_name keeps the EntityType, and en_DF_Date stores the completion time of newt Entity. For the Configuration Items that has only one aggregated events, EOF can be inserted under en_DF column. To arrange the data like this, we use python.

ent	en_ID	CI_name	CI_ID	SC	CI_type	max(date)	en_DF	en_name	en_DF_Date
Change	C00002217	SBA000716	5613	WBS000105	application	10/4/2013 12:00	C00002407	Change	10/7/2013 18:00
Change	C00002407	SBA000716	5613	WBS000105	application	10/7/2013 18:00	C00004088	Change	10/23/2013 15:00
Change	C00004088	SBA000716	5613	WBS000105	application	10/23/2013 15:00	C00005437	Change	11/6/2013 10:30
Change	C00005437	SBA000716	5613	WBS000105	application	11/6/2013 10:30	C00007175	Change	11/22/2013 15:00
Change	C00007175	SBA000716	5613	WBS000105	application	11/22/2013 15:00	C00007590	Change	11/28/2013 9:30
Change	C00007590	SBA000716	5613	WBS000105	application	11/28/2013 9:30	C00007748	Change	12/16/2013 9:13
Change	C00007748	SBA000716	5613	WBS000105	application	12/16/2013 9:13	C00006339	Change	1/2/2014 8:07
Change	C00006339	SBA000716	5613	WBS000105	application	1/2/2014 8:07	C00010017	Change	1/6/2014 18:00
Change	C00010017	SBA000716	5613	WBS000105	application	1/6/2014 18:00	SD0140382	Interaction	3/21/2014 16:51
Interaction	SD0140382	SBA000716	5613	WBS000105	application	3/21/2014 16:51	IM0045382	Incident	3/21/2014 16:51

Figure 4.11: en_DF,en_name, en_DF_Date columns for CI:5613.

It is also very crucial to decide how to calculate the predecessors and successors. We do not follow the same procedure as Hanser's.

One way to do it is calculating the Interactions and Incidents from the beginning until end of the path as can be seen from Fig.4.12.

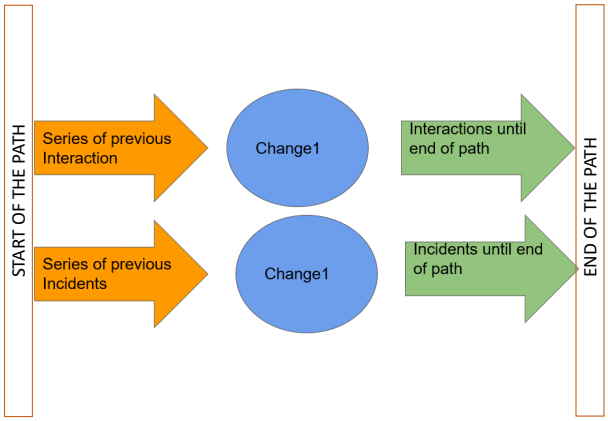


Figure 4.12: Calculating the interactions/incidents before and after a change

Another way is to calculate the Interactions and Incidents between the changes for every CI as can be seen from Fig.4.13.

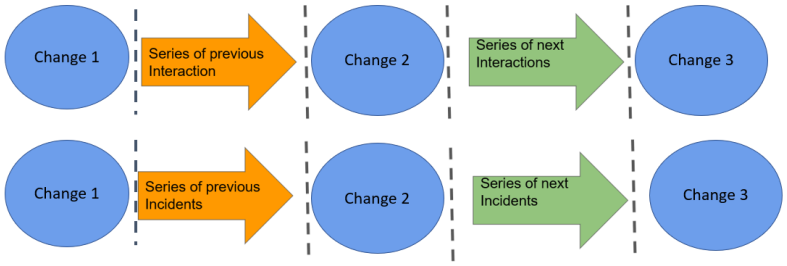


Figure 4.13: Calculating the Incidents/Interactions between Changes

For both options, Query 13 can be executed and the result in CSV files for each CI can be used to calculate the previous and next Interactions/Incidents. For Incidents and Change analysis, the filtering should be done only for Changes and Incidents. Interactions should be excluded. Same for Interactions. The filtering should be based on Interactions and Changes. Incidents should be avoided.

A very important question here is what if a path does not start with an Interactions/Incidents as Fig.4.14 illustrates. Then the Changes at the beginning can be ignored until the first Interactions/Incidents occur.

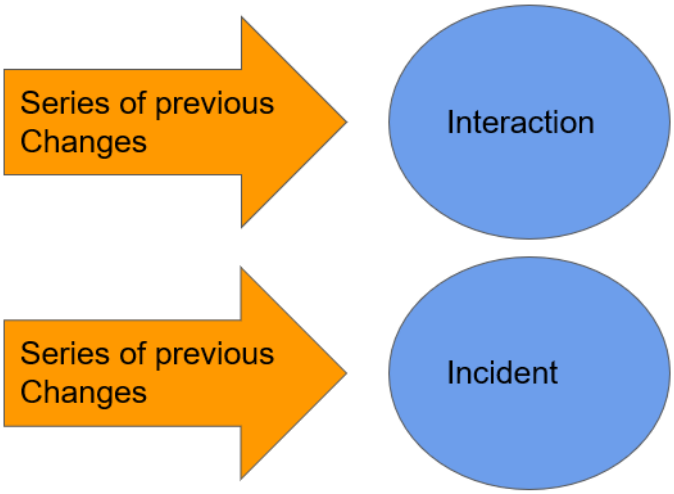


Figure 4.14: The paths starting with Changes. Ignore the first changes until finding the Incidents and Interactions.

4.5 Analysis Schema

As mentioned in the previous section we will order the entities depending on the last timestamp. We will have two different nodes. One of them is CI and the other is Entity. CI refers to Configuration Item, Entity refers to the new ordered Entities. In CI nodes we will have the following properties:

- id
- Name

For the Entities we will have:

- Entity Type
- Configuration Item ID
- Entity ID
- Originated From (Only for Changes)
- Completion Time

The following queries are constructing the nodes, uploads the record from CSV files for each CI, and create the relationships for (CI)-[:has]→ (Entity) and (Entity)-[:DF]→(Entity) First, we need to create the analysis schema with the Query 14 and Query 15.

```
//Query 14
CREATE CONSTRAINT ON ( ci:CI ) ASSERT ci.id IS UNIQUE
```

```
//Query 15
CREATE CONSTRAINT ON ( entity:Entity ) ASSERT (entity.id,
    entity.CI_ID) IS NODE KEY
```

A very important point here is that entities should have Entity id and Configuration Item id to be able to make them unique. Because one entity can belong to many Configuration Items and CI_ID makes every Entity unique.

We will also upload Originated_from property to under Change nodes as we use this information in the analysis. We can export this information with the following query.

```
call.apoc.export.csv.query{"MATCH
    (n:Entity{EntityType:"Change_Activity"}) RETURN
    n.Originated_from as Originated, n.IDraw as Changes",
    "Origins\_Change.csv"}
```

With the schema ready, we can now upload data from the CSV's for each Configuration Item with en_DF column.

Firstly, Query 16 should be run to create the Configuration Item nodes. Configuration Items will have id and CI_Name information. As a reminder, we mainly use id property for the queries.

```
//Query 16
LOAD CSV WITH HEADERS FROM "file:///file.csv" AS csvLine
MERGE(CI:CI {id:csvLine.CI_ID, Name: csvLine.CI_name})
```

Query 17 should be executed to create the Entity nodes which presents the aggregated events. The Entity nodes will have CI_ID, id, Completed information with related max(date).

```
//Query 17
LOAD CSV WITH HEADERS FROM "file:///file.csv" AS csvLine
MERGE(Entity:Entity{id:csvLine.en_ID,
    Name:csvLine.ent,CI_ID:csvLine.CI_ID,
    Completed:csvLine.max(date)
})
```

Query 18 should be executed to create the relationship "has" between

Configuration Item and Entity node, also en_DF relation between the aggregated events.

```
//Query 18
LOAD CSV WITH HEADERS FROM "file:///file.csv" AS csvLine
MATCH (CI:CI {id: csvLine.CI_ID, Name: csvLine.CI_name})
MATCH (Entity:Entity{id:csvLine.en_ID, Name:csvLine.ent})
MATCH (Entity_DF:Entity{id:csvLine.en_DF,
   Name:csvLine.ent_DF_name})
MERGE (CI)-[:has]->(Entity)
MERGE (Entity)-[:en_DF]- >(Entity_DF)
```

Finally, the following query should be executed to upload the Originated_from information. This record is necessary as Hanser is analysing the Origin of the Changes.

```
//Query 19
LOAD CSV WITH HEADERS FROM "file:///Origins_Change.csv" AS
   csvLine
MATCH (en:Entity {id: csvLine.Changes,
   Originated_from:csvLine.Originated})
```

Finally, Fig.4.15 shows the *Analysis Schema*.

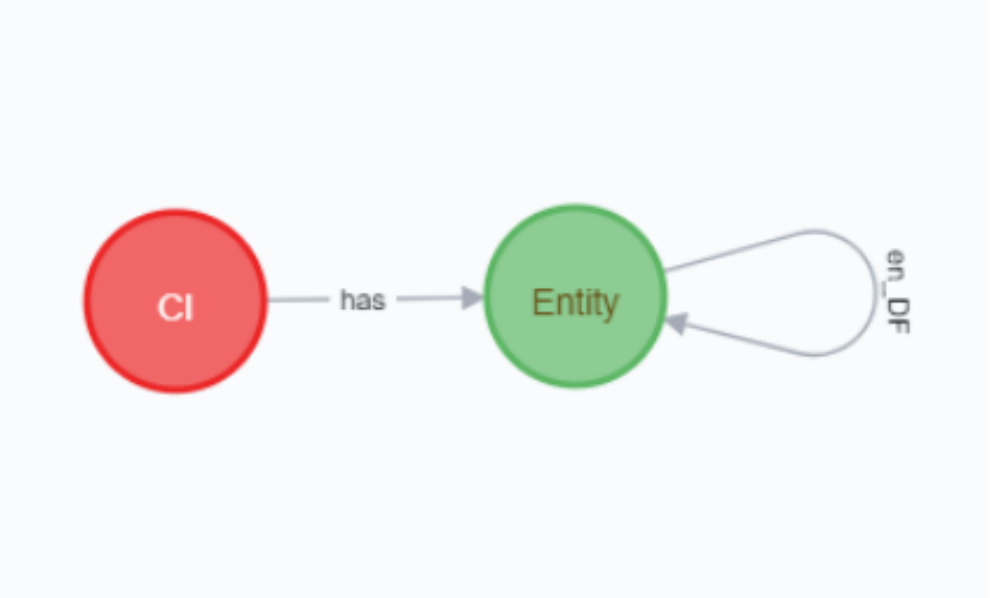


Figure 4.15: Analysis schema

Since the analysis schema is ready, we have to check with the amount of Change, Interaction, Incident entities with Leander's schema.

Table 4.2: Entity Comparison between Leander's schema and Analysis Schema

Entities	**Analysis Schema**	**Leander's Schema**
Configuration Item	14901	15864
Change	18000	18000
Incident	46606	46606
Interaction	147004	147004

Table 4.2 shows the amount of each entities in the two databases. The number of Change, Incident, and Interactions are same. However, the number of Configuration Item's number differ because 963 Configuration Items do not experience any Change, Incidents, Interaction in Leander's Schema. Hence, Analysis Schema does not contain 902 Configuration Items.

4.6 Analysing the predecessors and successors

In this section, we now analyze the number of interactions/incidents before/after a change. This implementation will not be same with Hanser's method. As mentioned in the previous chapter, there are two techniques to calculate the predecessors and successors. One way is, calculating the Interactions/Incidents before and after a particular change by ignoring the changes on the path. The second way is, calculating the Interactions/Incidents between two changes. When we apply the first way, we did not receive a very meaningful result as can be seen Appendix H. We therefore use the second option. The result can be seen on Table 4.3.

Table 4.3: Result of Second option

Entities	**previous Incidents/Interactions**	**post Incidents/Interactions**
CHG2	3	2
CHG3	2	2
CHG4	2	3
CHG 5	3	0

While calculating the preceding and following Incidents we ordered the aggregated Events by using Query 13 with only Changes and Incidents. Similar approach were used for Interactions, we filtered for only Changes and Interactions. Then we wrote a python code for calculating the precedings(prev) and followers(post) for each Change in each CI. Therefore, we calculated from the CSV's for each CIs. See Appendix B and C for the whole code.

We are assuming that this gives the more related information with the impact of Changes. Therefore, we will continue with the second option. The steps can be seen in Fig.4.16.

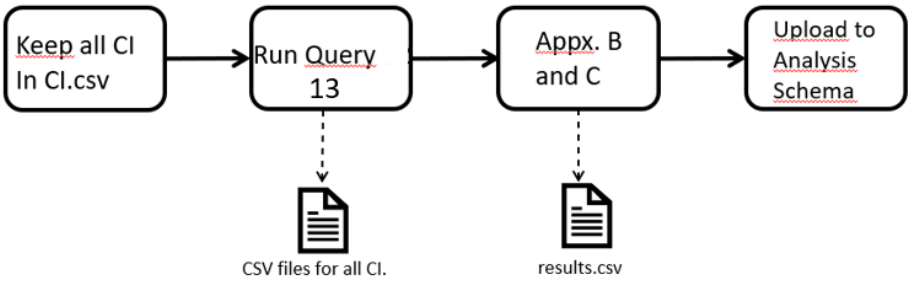


Figure 4.16: Impact Analysis Overview.

The information from the *results.CSV* file can be uploaded to the analysis schema we created. Query 20 can be executed to upload the number of previous incidents as "prev_inc" and Incidents until next change as "post_inc". Query 21 can be run to load "prev_int" for previous Interactions and "post_int" for next Interactions.

```
//Query 20
LOAD CSV WITH HEADERS FROM "file:///file.csv" AS csvLine
MATCH (Entity:Entity{id:csvLine.en_ID, CI_ID:csvLine.CI_ID})
set Entity.prev_inc=csvLine.prev_inc
set Entity.post_inc=csvLine.post_inc
```

```
//Query 21
LOAD CSV WITH HEADERS FROM "file:///file.csv" AS csvLine
MATCH (Entity:Entity{id:csvLine.en_ID, CI_ID:csvLine.CI_ID})
set Entity.prev_int=csvLine.prev_int
set Entity.post_int=csvLine.post_int
```

Both of these queries share same structure. The MATCH clause matching the entity nodes which aligns with the en_ID and CI_ID. Then SET clause is creating the aforementioned properties.

Query 20 creates 9248 properties. Query 21 creates 9358 properties. The number of Changes which experience previous and post Incidents is 4624 and 4679 for Interactions. In total we have 18000 Changes, and here we only have 4624 and 4674 Changes. The reason is while we calculate the predecessors and successors, we are looking for either Interactions or Incidents and Change to be in the path. To see how many of the traces are including Interactions or Incidents with Change we uploaded "log_graph.csv", from the result of Query 12, to ProM. We used "Explore Event Log" visualization. We found that more than 77% of the traces are experiencing only changes or only Interactions and Incidents as Fig.4.17 is presenting. Therefore, the many of the changes do not have the "prev_inc", "post_inc", "prev_int", "post_int" properties.

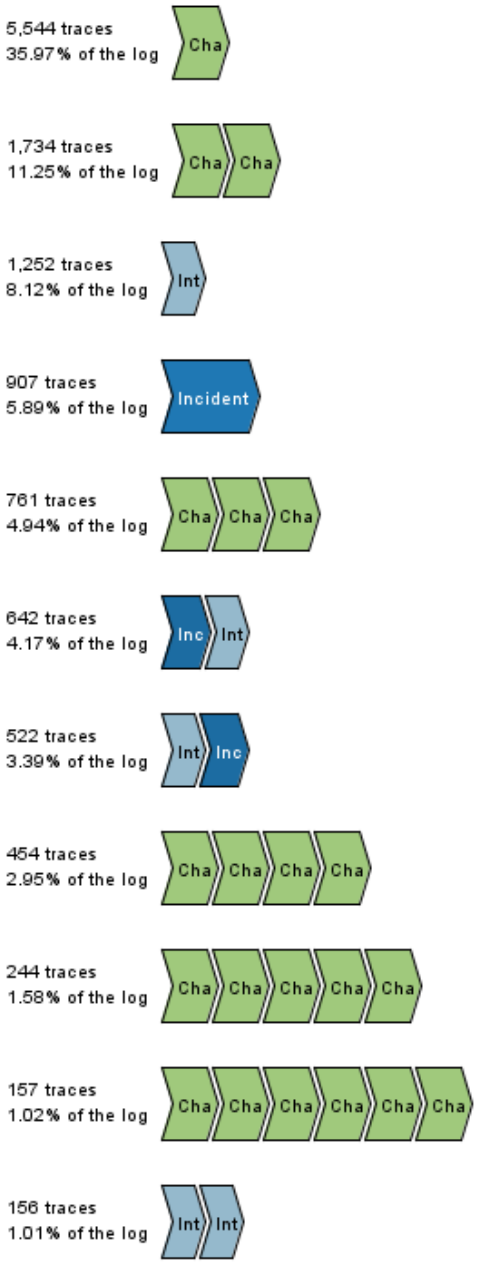


Figure 4.17: More than 70% of traces include only Change, Incident, Interaction.

We defined a new schema of aggregated events while keeping the relationship and presented a calculation of successors and predecessors of a change in order to conduct impact analysis.

This Schema will not only help us to conduct the same analysis with

Hanser but also, we create two different visualizations. In one of them, we create a visualization to detect the impact of Changes on multiple Configuration Item. This will show the path of each CI which is affected by multiple Changes. Then, we will cluster the Changes by color and see their impact for Interactions and Incidents.

Another schema is a scatter plot to understand better how the Configuration Items are mainly affected by Changes.

Chapter 5

Analysis Implementation

Under this Chapter, we introduce the methods to answer subquestions that Hanser proposed in his analysis. In Section 5.1, we introduce the queries for Subquestion 1.1 and 1.2. In Section 5.2, we introduce the queries for Subquestion 1.3 and 1.4

5.1 Analyzing Change Impact of Interactions

We will show the queries for Analyzing the impact of Change in terms of Interactions.

5.1.1 Detecting impact of Change

In our analysis we will concentrate on the analysis more on Changes perspective. We will look into the number of Changes which are followed by more interactions and less interactions. Query 22 shows the Changes which were followed by less interactions. The query is matching all the Change entities and returning the number of Changes which has less interactions afterwards.

```
//Query 22 - Finds the Negative Changes
match (en:Entity{Name:"Change"})
where toInteger(en.post_int)<toInteger(en.prev_int)
return count(en)
```

We also examine more about the causes of the Changes. To see the exact distribution of where the Changes are Originated from, we query the Originated_from property and their corresponding counts.

```
// Query 23 - Finds cause of Negative Changes
match (en:Entity{Name:"Change"})
where toInteger(en.post_int)<toInteger(en.prev_int)
return en.Originated_from as Origin, count(en.Originated_from)
    as Count
```

Query 24 shows the Changes which were followed by more interactions. The query is matching all the Change entities and returning the number of changes which has more interactions afterwards.

```
// Query 24 - Finds the Positive Changes
match (en:Entity{Name:"Change"})
where toInteger(en.post_int)>toInteger(en.prev_int)
return count(en)
```

To see the distribution of the Changes which are followed by more Interactions, Query 25 should be run.

```
// Query 25 - Finds the Origin of Positive Changes
match (en:Entity{Name:"Change"})
where toInteger(en.post_int)>toInteger(en.prev_int)
return en.Originated_from as Origin, count(en.Originated_from)
    as Count
```

On the other hand, it is also crucial to see how many changes are not followed by any interactions. It is very important here to compare prev_int and post_int because there are series of changes which are following each other and we would like to ignore these changes. We could not differentiate their impact which could be classified as neutral effect. Query 26 finds the changes which has less interactions afterwards with no interactions followed these changes.

```
// Query 26 - Finds the Changes which are not followed by any
    Interaction
match (en:Entity{Name:"Change"}) where
    toInteger(en.post_int)<toInteger(en.prev_int) and
    toInteger(en.post_int)=0 return en.Originated_from,
    count(en.Originated_from)
```

5.1.2 CI Analysis

It is also important to discuss how the Configuration Items are effected. Therefore, we can examine the number of Configuration Items which experience Interactions followed by Change eventually. Query 27 should be executed to find the necessary information. Before diving into Query 27, we can detail the steps.

Firstly, Query 27 filters the entities based on Interaction and Change and order the entities based on their Completed Time.

```
match (n:CI)--(en1:Entity)
where en1.Name="Interaction" or en1.Name="Change"
with  en1.id as id, en1.Completed as comp1, en1, n
order by en1.Completed
```

Then, collect the entity ids in a list and reads the every element of the list two by two as first and second.

```
with collect(id) as idList,n
unwind range(0, size(idList)) as i
with idList[i] as first, idList[i+1] as second,n
```

Afterwards, filters the elements based on first letter. If an entity id has 'S' as first letter then it means entity is an Interaction and if an entity starts with 'C' it means the entity is a Change. This filter means, one interaction will be eventually followed by a Change. Finally, it returns the number of Configuration Items that follows that rule.

```
where first starts with 'S' and second starts with 'C'
return count(distinct n.id)
```

Query 27 looks as follows.

```
// Query 27 - Finds the CIs that have Interactions followed by
   Changes
match (n:CI)--(en1:Entity)
where en1.Name="Interaction" or en1.Name="Change"
with  en1.id as id, en1.Completed as comp1, en1, n
order by en1.Completed
with collect(id) as idList,n
unwind range(0, size(idList)) as i
with idList[i] as first, idList[i+1] as second,n
where first starts with 'S' and second starts with 'C'
```

```
return count(distinct n.id)
```

We also check the number of Configuration Items which have Changes followed by Interaction Eventually. Query 28 should be executed to find this. It is very similar with Query 27 except the order of Change an Interaction. The first should be 'C' and second should be 'S'. Because we check Change followed by Interactions eventually.

```
// Query 28 - Finds the CIs that have Changes followed by
   Interactions
match (n:CI)--(en1:Entity)
where en1.Name="Change" or en1.Name="Interaction"
with  en1.id as id, en1.Completed as comp1, en1, n
order by en1.Completed
with collect(id) as idList,n
unwind range(0, size(idList)) as i
with idList[i] as first, idList[i+1] as second,n
where first starts with 'C' and second starts with 'S'
return count(distinct n.id)
```

5.2 Analyzing Change Impact of Incidents

Under this section, we introduce the Queries to answer subquestion 1.3 and 1.4.

5.2.1 Detecting impact of Change

Query 29 shows the Changes which were followed by less incidents. The query is matching all the Change entities and returning the number of changes which has less incidents afterwards.

```
// Query 29 - Finds the Negative Changes
match (en:Entity{Name:"Change"})
where toInteger(en.post_inc)<toInteger(en.prev_inc)
return count(en)
```

It is also necessary to obtain the causes of Changes. Query 30 shows the reasons and distribution of Changes in Incidents.

```
// Query 30 - Finds the Origin of Negative Changes
match (en:Entity{Name:"Change"}) where
    toInteger(en.post_inc)<toInteger(en.prev_inc) *
return en.Originated_from as Origin, count(en.Originated_from)
    as Count
```

Query 31 shows the Changes which were followed by more incidents. The query is matching all the Change entities and returning the number of changes which has more incidents afterwards.

```
// Query 31 - Finds the Positive Changes
match (en:Entity{Name:"Change"})
where toInteger(en.post_inc)>toInteger(en.prev_inc)
return count(en)
```

We also examine the reasons for Negative Changes. Query 32 gives the distribution of causes.

```
// Query 32 - Finds the Origin of Positive Changes
match (en:Entity{Name:"Change"}) where
    toInteger(en.post_inc)>toInteger(en.prev_inc)
return en.Originated_from as Origin, count(en.Originated_from)
    as Count
```

On the other hand, it is also crucial to see how many changes are not followed by any incidents. It is very important here to compare prev_inc and post_inc because there are series of changes which are following each other. We would like to ignore these changes as we could not differentiate their impact and they could be classified as neutral effect. Query 33 finds the changes which has less incidents afterwards with no incidents followed these changes.

```
// Query 33 - Finds the Changes which are not followed by any
    Incident
match (en:Entity{Name:"Change"}) where
    toInteger(en.post_inc)<toInteger(en.prev_inc) and
    toInteger(en.post_inc)=0 return en.Originated_from,
    count(en.Originated_from)
```

5.2.2 CI Analysis

The following queries are implemented to answer subquestion 1.4. We can see the number of Configuration Items which experience Interactions followed by Change eventually. Query 34, filters the entities based on Incidents and Change and order the entities based on their Completed Time.

Query 34 is very similar with Query 27 except we filter based on 'Change' and 'Incident'. The first letter of an Incident is 'I' and for Change 'C'. Since we check Incidents followed by Change eventually, the first should start with 'I' and second should start with 'C'.

```
// Query 34 - Finds the CIs which have Incidents followed by
   Changes
match (n:CI)--(en1:Entity)
where en1.Name="Incident" or en1.Name="Change"
with  en1.id as id, en1.Completed as comp1, en1, n
order by en1.Completed
with collect(id) as idList,n
unwind range(0, size(idList)) as i
with idList[i] as first, idList[i+1] as second,n
where first starts with 'I' and second starts with 'C'
return count(distinct n.id)
```

Query 35 finds the number of Configuration Items that have Changes followed by Incident eventually.

```
// Query 35 - Finds the CIs which have Changes followed by
   Incidents
match (n:CI)--(en1:Entity)
where en1.Name="Change" or en1.Name="Incident"
with  en1.id as id, en1.Completed as comp1, en1, n
order by en1.Completed
with collect(id) as idList,n
unwind range(0, size(idList)) as i
with idList[i] as first, idList[i+1] as second,n
where first starts with 'C' and second starts with 'I'
return count(distinct n.id)
```

5.3 Visualizing Impact of Changes on Multiple Configuration Items

Since we are working on graph databases, we gained more insights about the data. Query 36 finds the Changes which are effecting many Configuration Items at the same time.

```
// Query 36 - Finds the Changes affecting multiple CIs at the
   same time
// Match two Configuration Items and entities.
match (CI:CI)-->(en:Entity)
with CI, en
match (CI2:CI)-->(en2:Entity)
//Filter the entities as two same entity effecting two CIs
where en.id=en2.id and CI.id <> CI2.id
//Collect the CIs in a list and return the size of the list
   and the date that the Changes are effecting
return en.id as Changes, size(collect(distinct CI.id)) as CIs,
   collect(distinct en.Completed) as dates
//Return only the first 10 biggest one
order by CIs desc
limit 10
```

Applying Query 36 on our data revealed that indeed many changes are affecting multiple configuration .The top 10 largest changes affect 45-1006 CIs. Details are given later in Section 6.3. We implement a visualization to detect the Changes effecting multiple Configuration Items on Interactions and Incidents.

5.3.1 Detecting Changes Effecting multiple CI - Incidents

We want to show for each CI its sequence as a path of nodes of Incidents and Changes. To simplify the Visualization, we prefer to aggregate continuous sequences of Incidents into one node to avoid long sequences of interaction nodes.

We also want to color change nodes based on whether they have a positive impact or negative impact. Then, we show which changes work on two or more different CIs by connecting them with blue dashed line.

Finally, we also color the CIs depending on the impact that Changes which effect multiple CI. If a CI is green, that indicates it is effected mainly

positively, by the Changes which effect multiple CI. Red shows, CI is effected mainly negatively by the Changes which effect multiple CI. Salmon shows CI is effected neutrally which means the number of negative and positive Changes are same. We only share the main queries. For the implementation please refer to https://github.com/PinarTurkyilmaz/Event-Process-Mining-on-Graph-Databases/blob/master/CI_colored_Incident.py.

We run queries 37-40 to retrieve change nodes in the order in which they occur for a CI an we create the Visualization.

For each change node returned by query 37-40 we create a GraphViz node and create two interaction nodes (one before and one after the change node) with the number en.prev_inc and en.post_inc inside to indicate the number of interactions before/after the change.

We merged Incident nodes with prev_int property for each Change. Therefore, for each node we check whether prev_int is not 0, then we create it.

Query 37 should be run in the python script to receive the Positive Changes. This example is for Incidents. We are only interested in Changes because, we shrink the list of Incidents into one and only display the number of Incidents before implementing Change and after implementing Change.

```
// Query 37 - Receives the Positive Changes among the Changes
   affect multiple CIs (Incidents)
//Match two Changes
match (en:Entity{Name:"Change"}), (en2:Entity{Name:"Change"})
//Receive Positive Changes(en) and Negative Changes(en2), and
   check whether two are Changes are identical.
where toInteger(en.post_inc)<toInteger(en.prev_inc) and
   toInteger(en2.post_inc)>toInteger(en2.prev_inc) and
   en.id=en2.id
// Only take the Positive Changes
with distinct en
// Match the Configuration Items and the entities
match (CI:CI{id:en.CI_ID})- [h:has]->(g_en:Entity)
// Take only Changes.
where g_en.Name='Change' with g_en,CI order by g_en.Completed
//Collect the Changes into a list
with collect(distinct g_en) as nodeList, CI
// Unwind every element to build the relationship between
   Incident node and related Change node.
unwind range(0, size(nodeList)-2) as i
with nodeList[i] as first, nodeList[i+1] as second,
   last(nodeList) as third, CI
```

```
return distinct first, second, third, CI
```

Query 38 finds the Negative Effects of same Changes with Query 37 on different Configuration Items. We take the Negative Changes as the related Changes, *en2* to clearly see Change effect on the CIs. If we revisit the Fig. 1.1, *en* node here indicates the effect of CHG3 on CI2 as it has positive impact and en2 represents the effect of CHG3 on CI1.

```
// Query 38 - Receives the Negative Changes among the Changes
   affect multiple CIs (Incidents)
match (en:Entity{Name:"Change"}), (en2:Entity{Name:"Change"})
where toInteger(en.post_inc)<toInteger(en.prev_inc) and
   toInteger(en2.post_inc)>toInteger(en2.prev_inc) and
   en.id=en2.id
with distinct en2
match (CI:CI{id:en2.CI_ID})- [h:has]->(g_en:Entity) where
   g_en.Name='Change' with g_en,CI order by g_en.Completed
with collect(distinct g_en) as nodeList, CI
unwind range(0, size(nodeList)-2) as i
with nodeList[i] as first, nodeList[i+1] as second,
   last(nodeList) as third, CI
return distinct first, second, third, CI
```

5.3.2 Detecting Changes Effecting multiple CI - Interactions

We followed the same steps in Incident implementations. The detailed code can be found in https://github.com/PinarTurkyilmaz/Event-Process-Mining-on-Graph-Databases/blob/master/CI_colored_Interaction.py.

Query 39 finds the Positive Impact Changes for Interactions. Unlikely in the queries in Incidents, we used prev_int and post_int properties.

```
// Query 39 - Receives the Positive Changes among the Changes
   affect multiple CIs (Interactions)
match (en:Entity{Name:"Change"}), (en2:Entity{Name:"Change"})
where toInteger(en.post_int)<toInteger(en.prev_int) and
   toInteger(en2.post_int)>toInteger(en2.prev_int) and
   en.id=en2.id
with distinct en
match (CI:CI{id:en.CI_ID})- [h:has]->(g_en:Entity) where
```

```
   g_en.Name='Change' with g_en,CI order by g_en.Completed
with collect(distinct g_en) as nodeList, CI
unwind range(0, size(nodeList)-2) as i
with nodeList[i] as first, nodeList[i+1] as second,
   last(nodeList) as third, CI
return distinct first, second, third, CI
```

Query 40 finds the Negative Impact Changes for Interactions which is effecting multiple Configuration Items.

```
// Query 40 - Receives the Negative Changes among the Changes
   affect multiple CIs (Interactions)
match (en:Entity{Name:"Change"}), (en2:Entity{Name:"Change"})
where toInteger(en.post_int)<toInteger(en.prev_int) and
   toInteger(en2.post_int)>toInteger(en2.prev_int) and
   en.id=en2.id
with distinct en2
match (CI:CI{id:en2.CI_ID})- [h:has]->(g_en:Entity) where
   g_en.Name='Change' with g_en,CI order by g_en.Completed
with collect(distinct g_en) as nodeList, CI
unwind range(0, size(nodeList)-2) as i
with nodeList[i] as first, nodeList[i+1] as second,
   last(nodeList) as third, CI
return distinct first, second, third, CI
```

5.4 Classifying Impact of Changes on Multiple CI

We also create another visualization to see the how Configuration Items are effected mainly by the Changes which are effecting multiple Configuration Item.

5.4.1 Scatter Plot

As we can analyze changes which impact multiple CIs, we also want to assess whether changes have mainly a good impact on the majority of CIs involved or mainly a bad impact.

We crate a scatter plot which previous Interactions/Incidents on the x-axis and post Interactions/Incidents on the y-axis. We also colored the main

impact of Changes which are effecting more than one Configuration Items. If a Change has mainly bad effect on the Configuration Items, it will be visualized as **red**, if it has mainly positive impact then it will be in **green**, if the number of negative and positive impacts are same then it will be in **salmon**. We create separate plots for Negative and Positive impacts. We used matplotlib.pyplot package in python. The full implementation can be found in https://github.com/PinarTurkyilmaz/Event-Process-Mining-on-Graph-Databases/blob/master/scatter_plot.py.

5.4.2 Analysis on Interactions

For Interactions we run Query 41 and 42 first to be able to get Changes which has Positive and Negative Impact.

```
// Query 41 - Finds the Negative Changes with the number of
   Previous and Following Interactions
match (en:Entity{Name:"Change"}) where
   toInteger(en.post_int)>toInteger(en.prev_int)
return toInteger(en.prev_int) as
   prev_int,toInteger(en.post_int) as post_int, en.id as chg
order by prev_int desc
```

Query 42 finds Positive Changes based on Negative Change ids.

```
// Query 42 - Finds the Positive Changes with the number of
   Previous and Following Interactions
match (en:Entity{id:"%s"}) where
   toInteger(en.post_int)<toInteger(en.prev_int)
return toInteger(en.prev_int) as
   prev_int,toInteger(en.post_int) as post_int, en.id as chg
order by prev_int desc
```

Query 43 gives specifically Positive Changes and Positively affected Configuration Items. Query 44 gives specifically Negative Changes and Negatively affected Configuration Items.

```
// Query 43 - Receives list of CIs that are effected Positively
match (en:Entity{Name:"Change"}) where
   toInteger(en.post_int)<toInteger(en.prev_int)
return en.id as chg, collect(en.CI_ID) as good
```

```
// Query 44 - Receives list of CIs that are effected Negatively
match (en:Entity{Name:"Change"}) where
   toInteger(en.post_int)>toInteger(en.prev_int)
return en.id as chg, collect(en.CI_ID) as bad
```

To be able to decide how mainly a Change is effecting the Configuration Items, we collected the Positively and Negatively effected Configuration Items and the related Changes in different dictionaries such as positive_dictionary and negative_dictionary. We use Change ID as key, Configuration Item ID as value. Then, compared the amount of Configuration Items that same Changes have in both positive_dictionary and negative_dictionary.

5.4.3 Analysis on Incidents

For Incidents we follow the same pattern with Interactions, except prev_inc and post_inc are used. Queries between 44-46 are the main queries of the Classification. Query 45 and Query 46 gives Negative and Positive Changes respectively.

```
// Query 45 - Returns the Negative Changes with the number of
   previous and post Interactions
match (en:Entity{Name:"Change"}) where
   toInteger(en.post_int)>toInteger(en.prev_int)
return toInteger(en.prev_int) as
   prev_int,toInteger(en.post_int) as post_int, en.id as chg
order by prev_int desc
```

Query 46 finds Positive Changes based on Negative Change ids.

```
// Query 46 - Returns the Positive Changes with the number of
   previous and post Interactions
match (en:Entity{id:"%s"}) where
   toInteger(en.post_int)<toInteger(en.prev_int)
return toInteger(en.prev_int) as
   prev_int,toInteger(en.post_int) as post_int, en.id as chg
order by prev_int desc
```

Query 47 gives specifically Positive Changes and Positively affected Configuration Items. Query 48 gives specifically Negative Changes and Negatively affected Configuration Items.

```
// Query 47 - Returns the list of CIs that affected Positively
match (en:Entity{Name:"Change"}) where
   toInteger(en.post_int)<toInteger(en.prev_int)
return en.id as chg, collect(en.CI_ID) as good
```

```
// Query 48 - Returns the list of CIs that affected Negatively
match (en:Entity{Name:"Change"}) where
   toInteger(en.post_int)>toInteger(en.prev_int)
return en.id as chg, collect(en.CI_ID) as bad
```

To be able to decide how mainly a Change is effecting the Configuration Items, we collected the Positively and Negatively effected Configuration Items and the related Changes in different dictionaries such as positive_dictionary and negative_dictionary. We use Change ID as key, Configuration Item ID as value. Then, compared the amount of Configuration Items that same Changes have in both positive_dictionary and negative_dictionary.

Chapter 6

Is it possible to obtain the same results with the classical process mining techniques and tools?

6.1 Results of BPIC14

For this analysis, Hanser used Disco tools and for impact analysis they divided the main question into four subquestions, two for Interactions and two for Incidents.

- Subquestion 1.1: Is there in general a significant increase/decrease in the number of interactions after implementing Changes compared to before implementing Changes?
- Subquestion 1.2: Is there a significant correlation between the increase in number of interactions being related to PRODUCTS with Changes, after the Changes have been implemented?
- Subquestion 1.3: Is there in general a significant increase/decrease in the number of incidents after implementing Changes compared to before implementing changes?
- Subquestion 1.4: Is there a significant correlation between the increase/decrease in number of incidents being related to PRODUCTS with Changes, after the Changes have been implemented

Subquestion 1.1 Fig.6.1 shows the majority of Interactions, 98%, are taking place after implementing Changes. The blue area that contains case after 01.10.2013 06.47.

Figure 6.1: Interactions before and after Implementation of Changes

Hanser concluded that, for this question there is a strong correlation that implementing Changes leads to a significant increase in the number of Interactions.

Subquestion 1.2 They found that most of the Changes are not followed by a huge number of closed Interactions. See Fig6.2. Hanser states that this could be due to most Changes origination from Problems without related Interactions.

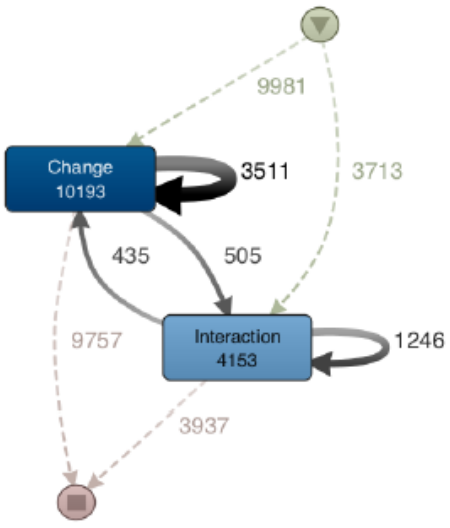


Figure 6.2: Changes and Interactions

Furthermore, they found that there are 44758 Interactions on 194 products which will eventually be related to a Change as can be seen from Fig.6.3.

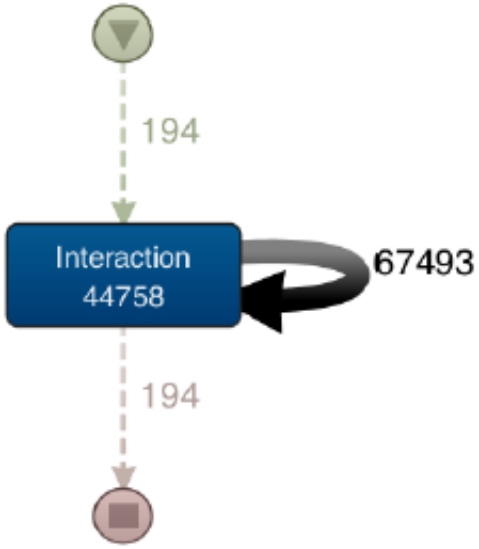


Figure 6.3: Interactions where the related product will subject for a Change

The number of Products which will eventually be caused for Interactions after a Change is 134 and the number of Interactions after implementing Change is 50668. Fig. 6.4 shows the detailed result.

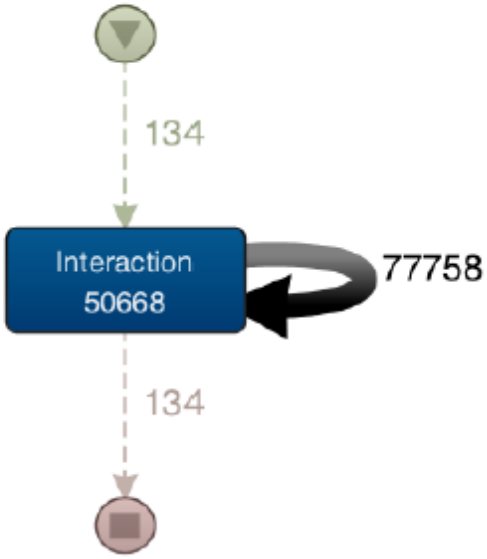


Figure 6.4: Interactions after implementing Changes.

The number of Interactions which follow related Changes (50668) is higher than the number of Interactions which subject Change (44758). There is an increase in number of Interactions by 5910(13.2%). Also, 194 products have Interactions before implementing Changes whereas 134 products have Interactions after implementing Changes.

When these findings were taken into account, Hanser defines implementing Changes as there is a decrease in the number of Interactions to a certain products whereas there is increase in other products. Furthermore, many cases with Changes are not followed by a huge number of Interactions which might be due to most of the Changes originate from Problem Management.

They found that most of the Changes are not followed by a huge number of closed Interactions. See Fig 6.2. Hanser states that this could be due to most Changes origination from Problems without related Interactions. 7

Subquestion 1.3 Fig.6.5 shows the majority of Incidents, 97% of all Incidents, are taking place after implementing Changes. The blue area that contains case after 01.10.2013 06.47.

Figure 6.5: Incidents before and after Implementation of Changes

Hanser concluded that, for this subquestion there is a strong correlation that implementing Changes leads to a significant increase in the number of Incidents.

Subquestion 1.4 They found that most of the Changes are not followed by a huge number of closed Incidents as Fig.6.6 shows. Hanser states that this could be due to most Changes origination from Problems without related Incidents.

Furthermore, they found that there are 11483 Incidents distributed on 177 products which will eventually be related to a Change as can be seen from Fig.6.7.

The number of Products which will eventually cause Incidents after a Change is 111 as can be seen from 6.8. The number of Incidents after the related change increases to 12241.

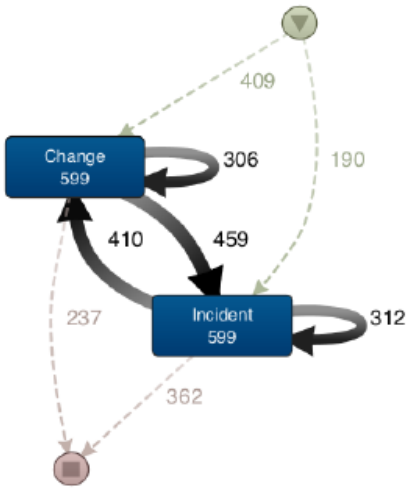


Figure 6.6: Changes followed by Incidents

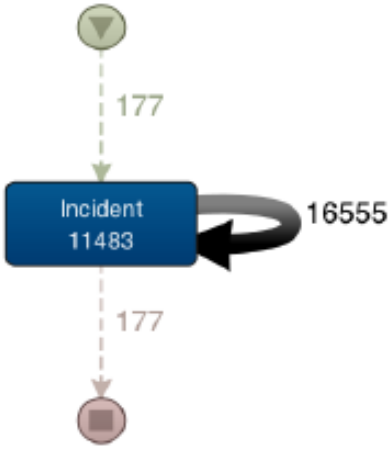


Figure 6.7: The number of Incidents where the related product will be subject for a Change

The number of Incidents which follow related Changes (12241) is higher than the number of Incidents which subject Change (11483). There is an increase in number of Interactions by 758(6.6%). Also, 177 products have Incidents before implementing Changes whereas 111 products have Interactions after implementing Changes.

When the findings were taken into account, Hanser concludes the impact

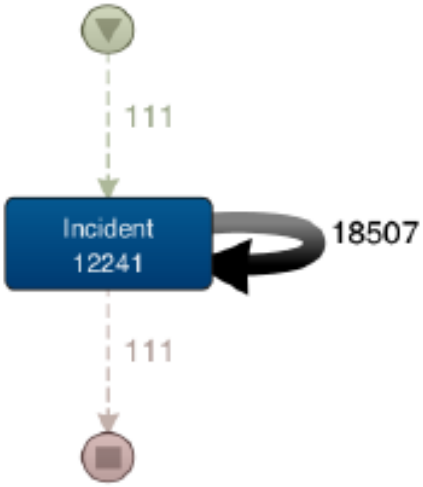


Figure 6.8: Number of Incidents after related Change Occurs.

analysis on Incidents as the increase in number of Incidents by 6.6% does not imply a strong correlation. Hanser also states that, this increase is not related to bad implementation of a Change and they recommend to examine if Changes causes bugs elsewhere in the software product.

6.2 Results of Impact of Change

6.2.1 Interactions and Change

Subquestion 1.1 Query 22 returns 1211 Changes. Table 6.2 shows the result of Query 22. As can be seen, 843 Changes are originated from a Problem whereas 369 Changes originate from Incident.

Table 6.1: Result of Query 22. Positive Changes are mainly Originated from Problems.

Origin	Count
Incident	369
Problem	842

Query 24 returns 996 Changes.

Table 6.2 shows the result of Query 25. As can be seen, 701 Changes are originated from a Problem whereas 293 Changes originate from Incident. There are also 2 Interactions which causes Changes.

Table 6.2: Result of Query 25. Negative Changes are mostly caused by related Problem.

Origin	Count
Incident	293
Problem	701
Interaction	2

The result of Query 26 shows 650 Changes are not followed by any changes.

To conclude, 479 of these Changes originates from Problem, and 171 of them originates from Incidents. There can be three reasons for these Changes not to followed by any Interactions.

1. These Changes are perfect changes and solved the problems, or;
2. The reasons that these Changes occur are related with other Problems not Incidents.
3. The Changes are taking place at the end of the data.

So far, we have 1211 Changes which have fewer Interactions than previous. These changes can be classified as "Positive Changes".

996 Changes have more Interactions than previous. These changes can be named as "Negative Changes". Since Positive Changes are more than Negative Changes, it is very possible to say that Changes have a good impact. However, there are 650 Changes which do not have any following Interactions.

Subquestion 1.2 is about how the Configuration Items are effected. Therefore, we can examine the number of Configuration Items which experience Interactions followed by Change eventually. Query 27 answers this question, and it returns 448 Configuration Items.

We also check the number of Configuration Items which have Changes followed by Interaction Eventually. Query 28 can be executed for that. It returns 468 CI.

When we check at the Configuration Item level, we see that more Configuration Item has Interactions followed by Change. Hence, on the Configuration Item level we can see that Configuration Items are effected mainly in negatively.

On the other hand, we found that the number of Positive Changes are more than the number of Negative Changes. This can be because many of the Changes are originated from Problem. Also, there are 650 Changes which are not followed by any Interactions.

To conclude, even though Positive Changes are higher than Negative Changes, the effect of Changes can be mixed for some Configuration Items. Some Configuration Items might have a good impact by some Changes where the others may be effected in a negative way.

6.2.2 Incidents and Change

We presented the results of Subquestion 1.1 and Subquestion 1.2. Now, we introduce the results of Subquestion 1.3 and 1.4.

The number of Positive Changes is 1271 as outcome of Query 29. When we analyze the root problem of the Changes, we see that 898 Changes originated from Problem whereas only 373 was originated from Incidents as outcome of Query 30. See Table 6.3.

Table 6.3: Result of Query 30. Positive Changes are mostly caused by Problems.

Origin	**Count**
Incident	373
Problem	898

There are 996 Negative Changes on Incidents from the outcome of Query 31. Table 6.4 shows that 290 of Negative Changes occurred because of Incidents, where 697 of Changes take place a related Problem. Only 2 of the Changes happened because of Interactions. This result is obtained by Query 32.

Table 6.4: Result of Query 32. Mostly Problems cause for Negative Changes.

Origin	Count
Incident	290
Problem	697
Interaction	2

We also executed Query 33 to see how many Changes are not followed by any Incident. Query 33 returns that 772 Changes are not followed by any Incident.

So far, we have 1271 Changes which have fewer incidents than previous. These changes can be classified as "Positive Changes". 772 Changes have more incidents than previous. These changes can be named as "Negative Changes".

Since Positive Changes are more than Negative Changes, it is very possible to say that Changes have a good impact. However, there are 772 Changes are not followed by any Incidents. The reasons for this are same with the previous analysis. It can be because of:

1. These Changes are perfect changes and solved the problems, or;
2. The reasons that these Changes occur are related with other Problems not Incidents.
3. The Changes are taking place at the end of the data.

The results are very similar with Interaction Analysis. Positive Changes are slightly higher than Negative Changes. 772 of Changes are not followed by any Incidents. Even though Negative Changes are mainly caused by Problems, still the number is less than the Problems which causes Positive Changes for both Incidents and Interactions. Therefore, it may worth to consider how a Change is effecting other Changes.

Subquestion 1.4 To answer Subquestion 1.4 which is about the Configuration Items are effected, we run Query 34. It returns 532 Configuration Items that have Incidents followed by change eventually. Additionally, Query 35 returns 559 Configuration Items have Changes followed by Incidents eventually.

The results are very similar with analysis on Interactions. When we checked at the Configuration Item level, we see that more Configuration Item has Incidents followed by Change. Therefore, on the Configuration Item level we can see that Configuration Items are effected mainly in a negative way.

To conclude, even though Positive Changes are higher than Negative Changes, the effect of Changes can be mixed for some Configuration Items. Some Configuration Items might have a good impact by some Changes where the others may be effected in a negative way. Finally, it is also worth to examine further about the effect of Changes between each other as even though Negative Changes are occurring because of Problems.

6.3 Visualization Analysis

The result of Query 36, which finds that Changes affect multiple CIs at the same time. The result can be seen in Fig. 6.9.

"Changes"	"CIs"	"dates"
"C00001707"	1006	["2013-10-13 07:00:00"]
"C00001334"	122	["2013-11-01 18:00:00"]
"C00000480"	109	["2014-03-28 18:00:00"]
"C00003258"	73	["2013-10-16 18:00:00"]
"C00013425"	68	["2014-02-07 00:00:00"]
"C00011096"	67	["2014-01-22 08:29:00"]
"C00001240"	60	["2013-12-14 12:00:00"]
"C00000745"	55	["2013-10-04 17:00:00"]
"C00003046"	45	["2013-11-29 18:00:00"]
"C00006164"	45	["2013-11-18 18:37:00"]

Figure 6.9: Changes affecting multiple CIs at the same time.

This result gives us the idea to see the impact of Changes on multiple CIs. Details are provided in the further sections.

6.3.1 Interactions and Changes

Appendix D shows the effect of Changes which affect multiple CI for Interactions.

In total we have 35 Configuration Items, 5 of them are affected neutrally by the Changes which effect more than one product. 15 products are effected negatively by the changes and 17 products are affected positively.

If we look in more detail we will see different cases for different Changes and Configuration Items. C000011104 is effecting 3 products which has ID 3069, 3057 and 927 as can be seen on Fig. 6.10. For 3057, C000011104 has a positive impact because it did not lead to any Interaction afterwards. However, when C000011104 is implemented on 927 and 3069, it increases the Interactions afterwards hence leads to a negative impact. These number of Interactions are the peak that 3069 and 927 experiences.

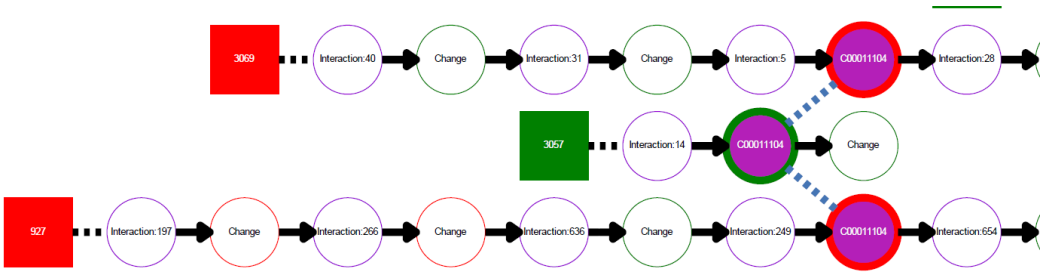


Figure 6.10: C000011104 is effecting 3 Configuration Items. On two of them, it has mainly bad effect.

C00001688 which is the most complex one in the picture. This Change effects 6 different products which are 750, 477, 853, 125, 469, 15567 and mainly it has good effect. Fig.6.11 shows the IT Components that C00001688 effect. The CI ids are on the changes and their colors indicating the color of the relevant CI.

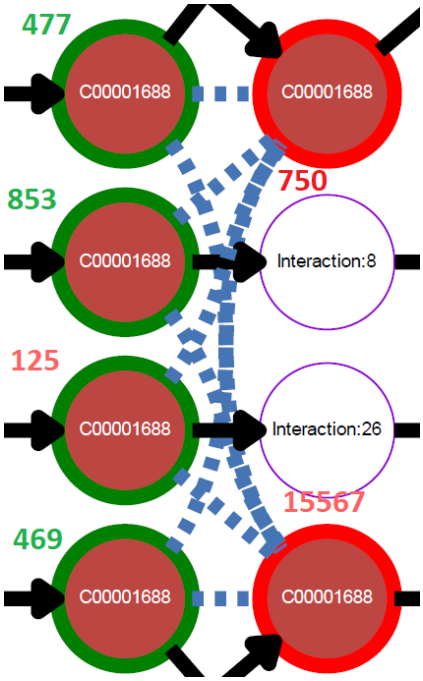


Figure 6.11: C00001688 is effecting 6 products and mainly has a good impact.

Another interesting Change is called C0003946. It is effecting 5 Configuration Items which are 853, 125, 469, 15562, 787. On the products, C00003946 has mainly negative impact as can be seen in Fig. 6.12.

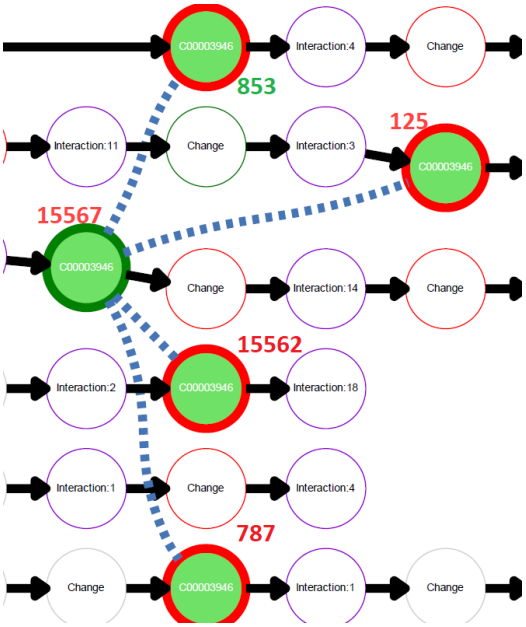


Figure 6.12: C00001688 is effecting 5 products and mainly has a negative impact.

C00003946 is effecting 3 of the same products, which are 853, 125, 15567, that C00001688 is effecting. C00003946 has always the reverse impact of what C00001688 as can be seen on Fig.6.13.Even though 853 and 125 are affected positively by C00001688, C00003946 has a negative impact on these Configuration Items.

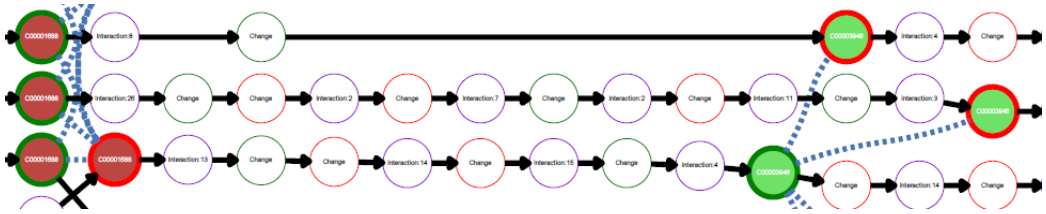


Figure 6.13: C0003946 has the reverse impact of C00001688 on the products.

Furthermore, if we look at the general picture, when there is a good change which is effecting more items, usually the change right after is having a bad or neutral effect. Very few of them are followed by a positive change. This finding supports the result in the previous analysis and lead us ask the same question: "How do changes are effecting each other?" .

6.3.2 Incidents and Changes

Appendix E shows the impact on Incidents.

There are 41 products in Appendix E. 5 of the Configuration Items have neutral effect, 17 of them has negative effects, and 19 IT components are effected positively. There can be different Configuration Items when compared to the Interaction pictures. This is because maybe the changes does not have any negative or positive impact on that Configuration Items in terms of Interaction. Therefore, the effect of same Changes may differ for different implementations, e.g. Incidents and Interactions.

There are three Configuration Items 5617,5637,5638, which has related Change, C00005777. C00005777 has positive impact on 5637 whereas negative impact on 5638. Right after C00005777, these two Configuration Items experience C00006732 which exactly reverse effect of C00005777. 5637 is experiencing another Change called C00008489 afterwards of C00006372. The new change is shared with 5617, and C00008489 has positive impact on 5637 after C00006372. The whole path can be seen on Fig.6.14.

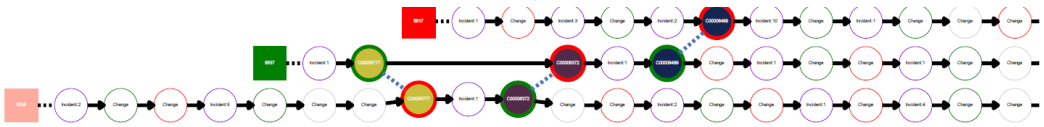


Figure 6.14: C00006372, C00005777, C00008489 on 5617,5637,5638.

C00001688 has a very similar effect when the analysis conducted on Incidents as well. For Incidents, the change is effecting 7 products. 6 of them are same with Interaction analysis. Additional Configuration Item is 8056 and this Change has a positive impact on 8056. See Fig.6.15 for more details.

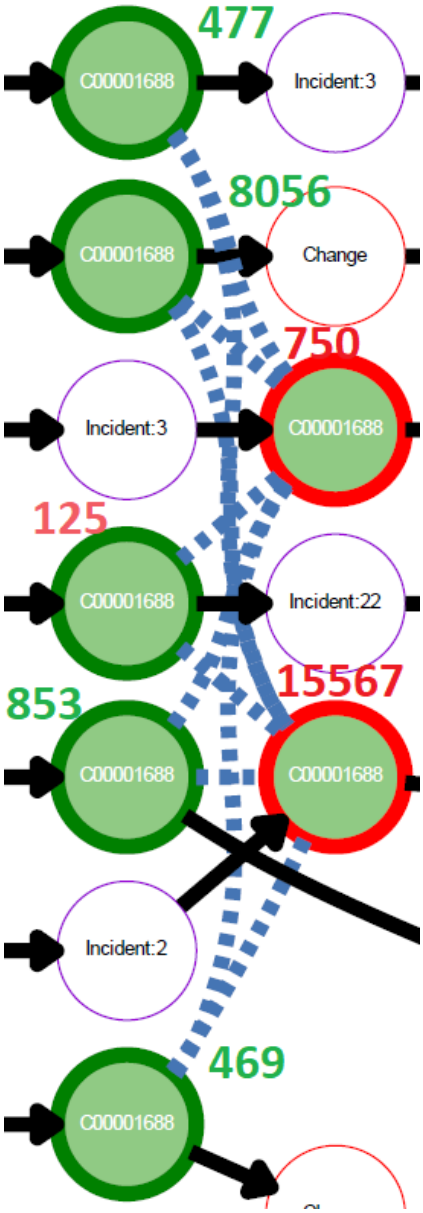


Figure 6.15: C00001688 is effecting 7 Products and mainly has a positive impact as it is in Interaction Analysis.

We do not see this Configuration Item on the Interaction analysis C00001688 does not have any positive or negative impact on this Configuration Item in terms of Interactions.

The effects of C00003946 for Incidents in Fig.6.16 , it has mainly negative impact on the products. The products it is effecting are same with the Interaction analysis.

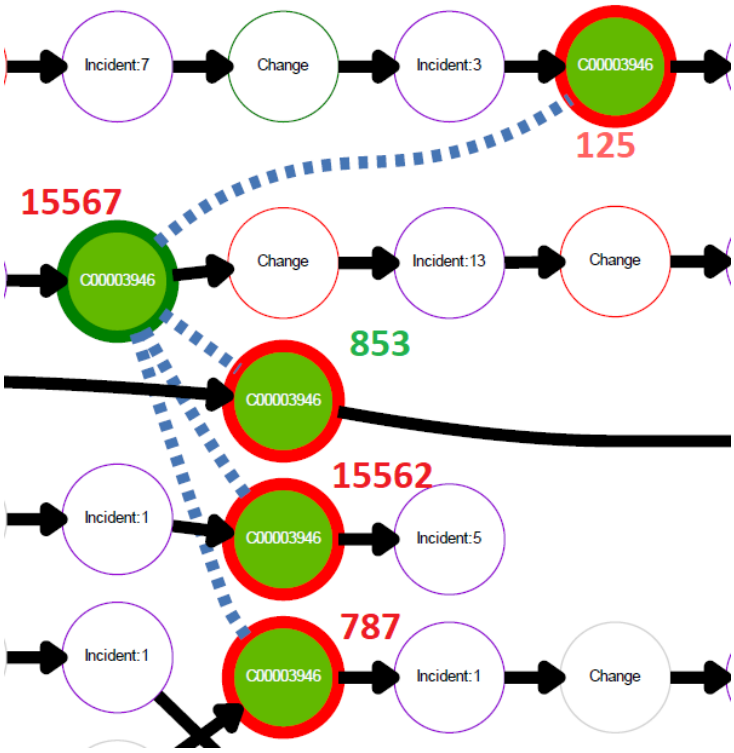


Figure 6.16: C00003946 is effecting 5 Products and mainly has a negative impact as it is in Interaction Analysis.

The relation between C00001688 and C00003946 is very similar as it was in Interaction analysis. 3 of the products, which are 15567, 125, 853, are experiencing both of the aforementioned Changes, first C0001688 and eventually C00003946. For product 125 and 853, C00001688 has positive impact where C00003946 has the negative impact. 15567 is experiencing the reverse effect of 125 and 853.

A very interesting result we obtain was about Configuration Item 15567. We could also see that IT component 15567 is experiencing negative impact on the Interaction analysis whereas it has neutral effect on Incident perspective. The reason is 15567 is experiencing two more Changes, which are C00015024 and C00012061. These Changes effect another Configuration Items. Fig. 6.17 shows that C00012061 eventually follow C00015204 on 15567, 850 and has Negative and Positive Impact respectively. 853 experience a negative impact from C00012061.

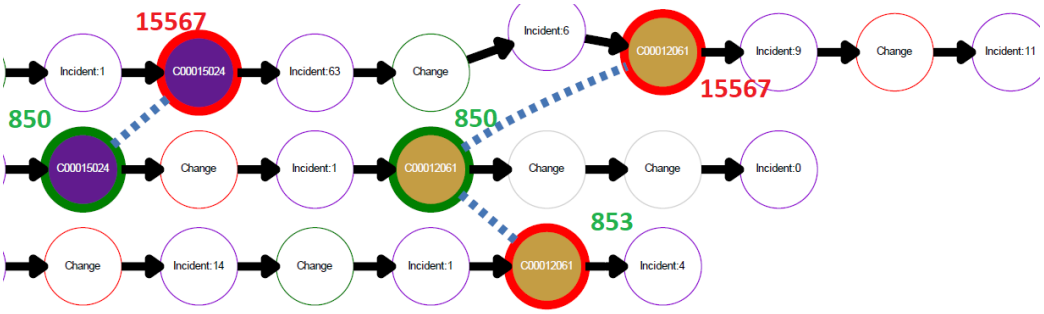


Figure 6.17: C00012061 eventually follow C00015204 on 15567, 850 and has the same impact.

To conclude, Changes are effecting multiple Configuration Items with a mixed effect. One change can have positive impact on one Configuration Item and negative impact on others. Moreover, the configuration items can be effected differently when analysed for Interactions and Incidents. Moreover, we see that Positive Changes which effect multiple Configuration Item are usually followed by a Neutral or a Negative Change. With the help of this visualization it was possible to see the impact of Changes on different products.

6.4 Results Classifying impact of changes on multiple CIs

6.4.1 Interactions and Changes

Fig.6.18d shows the Positively Effected Interactions and Fig.6.18c shows how are the Configuration Items are effected mainly.

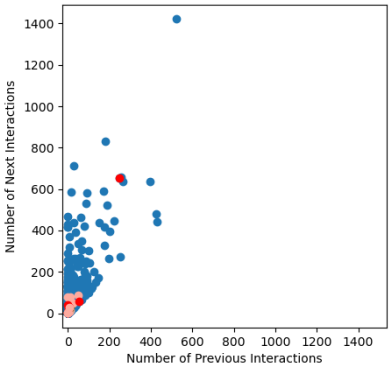
Even though Changes are in the Positive impact table, the Configuration items are effected equally negative which resulted as neutral effect for the Changes. Moreover, the Changes which effect more than one Products do not have extreme effects. As expected, there are also Changes on Positive table which have mainly positive effects. Finally, a few of the Changes are mainly effecting the Configuration Items as negatively.

The Negative Changes for Interaction analysis can be seen on Fig. 6.18a and Fig.6.18b

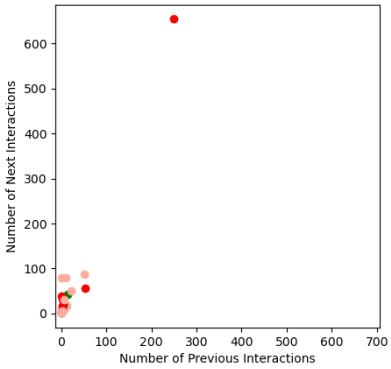
We see that the Configuration Items are effected negatively on extreme cases. Unlike the Positive Impact table, in Negative Impact table there are Changes which results 1400 Interactions after 800 Interactions. When we examine the Changes which effect multiple Configuration Items, we see that

there are extreme cases such as resulting more than 600 Interactions after implementation.

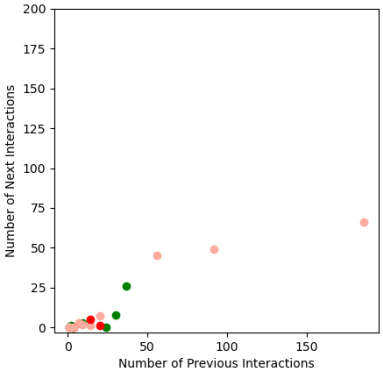
Moreover, the Configuration Items are effected either neutrally or negatively. There are a few Changes which effect products mainly positively even though they take place in the Negative impact graph.



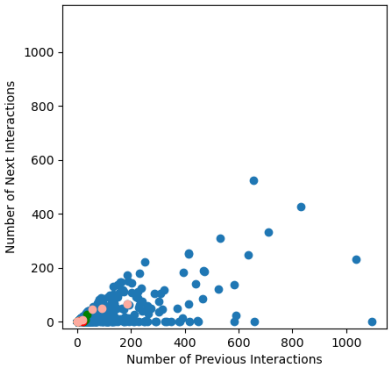
(a) Negative Changes- Interactions.



(b) Negative Changes-Interactions (Detailed).



(c) Positive Changes - Interactions (Detailed)



(d) Positive Changes - Interactions

Figure 6.18: Plots of Interaction - Change

6.4.2 Incidents and Change

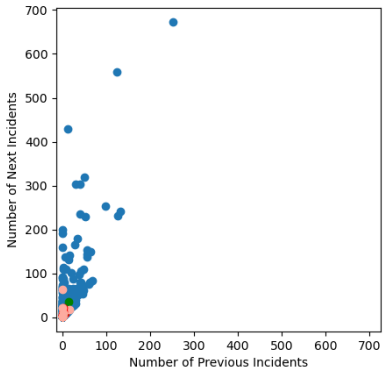
When we see Fig. 6.19d, we can say that Changes are mostly decreasing the number of Incidents in half. When we examine Fig. 6.19c we see that there are no extreme cases for the Configuration Items which are effected by multiple Changes. Mostly, Configuration Items are effected either positively

or equally negative; which results as neutral. There are a few mainly Negative Changes even though they effect the some of the Configuration Items positively.

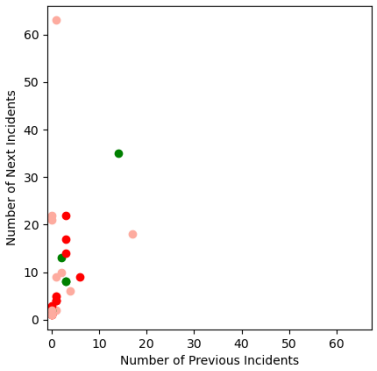
Fig.6.19a and Fig.6.19b show the Negative impact Changes and impacts on the Configuration Items respectively.

Fig.6.19a shows that, Changes mostly double the number of previous Incidents. Fig.6.19b shows that when the negative Changes effect multiple Configuration items, they mainly have either neutral or bad effect. There are also Configuration Items which are effected mainly positive by some of the Negative Changes.

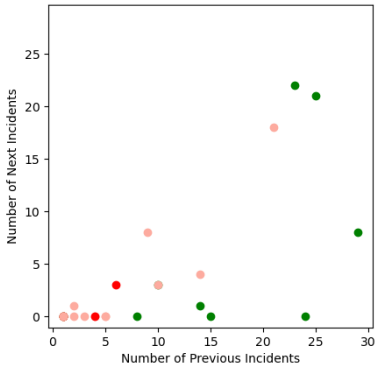
Even when we compare our results between Incidents and Interactions, we see a different effect. On Interactions it was possible to see the extreme effects of Changes on Configuration Items which we do not obtain for the effect on Incidents. Furthermore, we also show that, Changes can have multiple effects even though they are considered positive or negative on one Configuration Items.



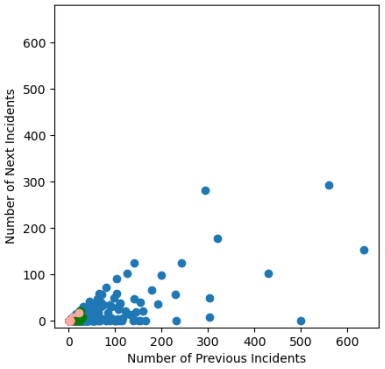
(a) Negative Changes- Incidents.



(b) Negative Changes-Incidents (Detailed).



(c) Positive Changes - Incidents (Detailed)



(d) Positive Changes - Incidents

Figure 6.19: Plots of Incident - Change

6.5 Comparison

Method The data pre-processing for the analysis has been conducted differently. On the BPIC 14 report, they mainly concentrate on Incident and Interaction perspective. They took the series of changes as one change. Hansen stated this implementation as *Implementing Changes* is best represented as the collection of changes, because a single Change followed by another Change will be considered one major Change by the users – thus the Interactions and Incidents placed after the Changes will be impossible to distinguish if belonging to the first or second Change. It is assumed that "A Change" is the collection of Changes.". Fig.2.5 shows Hansen's method for Interactions.

We did not execute that processing in our analysis. We kept every change as in their order and calculated the successors and predecessors between Changes. The reason we did not follow Hansen's method is because if the Change occurs in a long term, it should not be included in one list. Because that could result in losing the information and interpreting wrongly for that particular change. That's why, Hanser's perspective is underestimating the Changes and mainly concentrate on Interactions/Incidents. Since we did the processing differently we did not obtain the same numbers with their results.

We also did not remove the Changes which does not have *Actual_End* the number of Changes which does not have *Actual_End* activity is very low, 1311 out of 18000 Changes. We do not think it will effect the results. In this analysis, it is assumed the incomplete Changes do not have to be filtered out as the number is pretty low.

In terms of Configuration Item level, they removed the cases which does not have Change as the first activity. However, we did not conduct this method, instead we trimmed until the first interaction. Therefore, the numbers are higher and the results are different.

The preprocessing for the Incidents also took place in the same way.

Subquestion 1.1 For Interaction Analysis, we found that Changes has mainly less Interactions than previous. On the other hand, Hanser found that implementing Changes increases the number of Interactions as all the Interactions start after the start date. However, the approach here is very simple because it is expected that all the Interactions are going to take place after the start date. Therefore, we do not have the exact same result.

Subquestion 1.2 We found that the Configuration Items which has Interactions eventually followed by Change are less than Configuration Items that has Changes eventually followed by Interactions. Therefore, we state that the changes might not be that helpful. However, Hanser concluded that the number of products decrease before implementing Change and after implementing Change.

Furthermore, we found that many Changes are not followed by any Interactions and there can be three reasons for that; the change was very helpful or the change is originated by a problem instead of a related Interaction or maybe that Change is already at the end of the data. Hanser is also stating this issue, and reasoning this point as the Changes are mainly originated by a problem.

We both concluded that, there are no strong correlation between Interactions and Changes. Implementing Changes can decrease the Interactions

for some products and decrease for other products.

The result for Incidents are very similar.

Subquestion 1.3 For Incidents Analysis, we found that Changes has mainly less Incidents than previous. On the other hand, Hanser found that implementing Changes increases the number of Incidents as all the Incidents start after the start date. As mentioned before, our results are not similar because of the different approach we applied.

Subquestion 1.4 We found that the Configuration Items which has Incidents eventually followed by Change are less than Configuration Items that has Changes eventually followed by Incidents. Therefore, we state that the changes might not be that helpful. However, Hanser concluded that the number products decrease before implementing Change and after implementing Change.

Additionally, we found that many Changes are not followed by any Incidents and there can be three reasons for that; the change was very helpful or the change is originated by a problem instead of a related Incident or maybe that Change is already at the end of the data. Hanser is also stating this issue, and reasoning this point as the Changes are mainly originated by a problem. However, we should remind that the Negative Changes are not meant to mean the bad quality of Changes, instead implementing Changes can result in a need of many Changes in the system.

Conclusion We both concluded that, there are no strong correlation between Incidents and Changes. Implementing Changes can decrease the Incidents for some products and decrease for other products.

In order to sum up, since we are working on the changes perspective, as the data was processed by the Change angel. Hence, the numbers that they are suggesting and we are finding are not matching. Therefore, the unequal results are predictable and expected. However, the main result that we obtained are same:

- There is no unique result that changes have good or bad effect on Interactions as the impact can vary between the configuration items.
- There is no unique result that changes have good or bad effect on Incidents as the impact can vary between the configuration items.

6.6 Summary

Our general result with Hanser is mainly similar, even though in detail there are some different approach.

With the help of additional findings, we found more details insights about the impact of Changes. We found Changes effect multiple Configuration Items at the same time. Based on this information, we implemented two different visualizations, one of them shows the aggregated events in order, from perspective of Changes. We found that the impact on Configuration Items can vary between Interactions and Incidents. Some Changes would not have any Negative or Positive effect on Configuration Items, and that would change the result for that particular Change and Configuration Item.

The second one shows where the Changes which effect multiple Configuration Items sit when compared to whole Changes. We saw impact of Changes can vary between the Configuration Items. Configuration Items can mainly be affected positively even though the related Change has Negative Impact for other Configuration Items. Furthermore, we found that in terms of Interactions and Incidents, the Configuration Items can have different experience of Changes. Moreover, Changes which effect multiple products mostly are not pictured in the extreme cases. With process mining techniques this would not be identified but thanks to graph-based structure, which helped us to see the whole path for an IT component, we had more insights about the effects of changes.

Chapter 7

Lessons Learned

We would like to discuss how the queries can be generalizable on different dataset.

The following practices can be generalizable as they can be conducted in any eventlog. We will mention the general structure and exclude the unique entity names which would change depending on the dataset.

- Projection onto a subset of events (creating relationship between not DF event nodes):
  We had to deal with the system generated event nodes. This exercise could be very useful in case the nodes are meant to be kept. Query 9 is the query where we can create a relationship which excluding some of the event nodes. The general structure is as follows:

```
MATCH the related path
WHERE if any filtering is needed on the Entities, Events,
   etc.
WITH if needed take desired properties
WHERE NOT undesired nodes
WITH collect(desired nodes) as list
UNWIND RANGE(0, size(list)-2) as i
WITH list[i] as first, list[i+1] as second
MATCH the nodes
WHERE the ids are same
CREATE the relationship
```

- Remove erroneously recorded event nodes:
  In case of any duplicates, Query 10 can be executed to delete the nodes without needing any specific information in the dataset. We call the information we want to delete as *nodes* The general structure is as follows:

```
MATCH the desired path
WITH the ids which make the path unique,
   collect(nodes) as list
WHERE size(list)>1 to check whether the information
   is duplicated
FOREACH (n in tail(list) | DETACH DELETE n) deletes
   everything for the duplicated nodes except the
   head. Leaves only one node.
```

- Aggregating of sequences of events into one single event:
  It is also very crucial to discuss the general structure of aggregation of the consecutive events into one aggregated event based on the last timestamp and show the Entity Type that they belong to. Then the new aggregated Events will be named as the Entity Type that they belong to. Query 12 can be referred back to obtain the general structure of aggregation.

```
MATCH the related path
WHERE the desired property is given
WITH the ids and types which make a path unique also
   with end date of events
ORDER BY the end date
WITH the desired properties (could be same with the
   previous WITH)
WHERE NOT in case any 'null' property occurs because
   of filtering
RETURN the wanted information, max(end date)
ORDER BY max(end date) to start order the aggregation.
```

- Finding eventually Directly Follows events/entities:
  Finding the eventually follows relation can also be very useful. One way to execute this is using [*] in the queries as relation. This operator allows to obtain a graph pattern where the number of hops between nodes is not known. However, if the dataset is big, this operation takes

excessive time to answer. We have to write the efficient method to execute this operation. Therefore, Query 27 is the best way to obtain the results in the most possible fast way. The general structure can be as follows:

```
MATCH the corresponding path
WHERE the desired nodes to see the eventually DF
WITH the ids and last time stamp
ORDER BY the last timestamp
WITH the list of node properties where it is easy to
   differentiate, e.g. collect(id) as list
UNWIND range(0, size(list)) as i
WITH the elements in consecutive order, e.g. list[i]
   as first, list[i+1] as second
WHERE the desired condition matches for first and
   second
RETURN the eventually DF nodes, e.g. RETURN first,
   second
```

Even though there are queries which results fast, the Neo4j takes too much time to render if the path is enormously long. Therefore, another tool can be necessary to visualize in a quick and efficient way.

Since Neo4j is also a graph, it is very easy to grasp the idea of GraphViz. GraphViz is a package in python which is very useful and easy to implement if any visualization is necessary on top of graphs. There are also many points that can be discussed and make the implementations easier.

A very useful insight which gained during the implementation of Visualization is, if there exists a long path, even GraphViz is not enough to render.

To overcome this problem, it is possible to concentrate only on one perspective. This means that keeping the nodes that mainly was concantrated on and show the numbers of how many times the other nodes occurs. In our case, we had the perspective of Changes so we just showed number of previous or followed Incidents/Interactions for the related Change.

With the help of this idea, even though there are huge number of predecessors or successors of a Change, with one node it is still possible to keep the insight from the event log.

Chapter 8

Conclusions

In this thesis, we discussed how to conduct a process mining analysis on graph event log. More specifically we answered how a schema should be to make a proper process mining analysis. We also discussed about the data preparation in a graph schema for such analysis. Then we created new schema to conduct the analysis. To validate our results, we compared our findings with a classical process mining techniques applied. Finally, we developed two different visualization to grasp the graph competency of graph structure and we obtained more detailed results.

We had two event schema on graph databases about incident management system by Esser and Leander and we conducted an impact analysis. We first give insights about both schemas and discussed which of them is more suitable for process mining analysis. Esser's schema was easy to interpret but did not have a complete DF relation and a concrete activity definition of events. Leander's schema was harder to understand but it has a complete DF relation and clear event definition. Even though the queries take too much time and intense filtering is necessary, we decided to continue with Leander's schema.

We needed to prepare the data for the analysis. There were events that needed to remove, either because there were duplicated records or system generated records which is not necessary for analysis to keep the information. We found two different solutions, one of them was to create a new relationship called DF_Mat by excluding the undesired event nodes and follow this relationship. Meanwhile, the other one was deleting the nodes and creating the DF relationship again. We decided to continue forward with the latter one as the first solution would only be valid for system generated event nodes and not for the duplicated records. Moreover, it would not be very easy to query with the new relationship afterwards as always a filtering would be needed for the relationship.

After preparing the data we created another schema, so called Analysis Schema, by aggregating the event nodes into one but keep the relationship between them. At the end we had the path for each Configuration Item with aggregated events in order.

Then for the impact analysis we had to calculate the previous/next Incidents and Interactions of a related Change. First we calculated it since the beginning of the path to a related Change and until end of the path from a related Change. However, it did not give us a very detailed information as we did not know which Incident/Interaction belong to which Change. Therefore, we calculated the Incidents/Interactions between the Changes. That gave us more elaborated results. We uploaded the results to the Analysis Schema.

Then we conducted analysis and compared the results with an analysis where classical process mining tools and techniques were used. Our processing was slightly different then they applied because in their processing, an information loss could be possible.

When we compare our analysis we found that our results in detail were not same because of the difference in processing of the data. However, our general conclusion is very similar. They found that effect of Changes can vary between Configuration Items. We also found the same result when we conducted the same analysis.

Additionally, we found the unique Changes that effect multiple products and obtained very elaborated insights. Firstly, one Change can be Negative for one product and Positive for another product. Secondly, in terms of Incidents and Interactions, Configuration Items can be impacted differently. Finally, Changes which effect multiple products mainly are not pictured in the extreme cases. They mostly have very minor effects on the Configuration Items. To find these insights, classical process mining would not be enough.

8.1 Limitations and Future Work

First of all, our analysis was not very detailed process mining analysis. We defined techniques to find whether a schema is applicable, write efficient queries, and feature extraction from data. However, a deep process mining analysis is missing. Therefore, future work could look into the implementation of different process mining algorithms and extracting the original process model by using the queries and the ideas presented in this report.

Secondly, we were able to finish impact analysis on Incidents and Interactions. The rest of BPIC14 is still remaining. The Analysis Schema can be used to analyse how much time does it take for a product to return to a steady state. Future work can contribute to finish the BPIC14.

Thirdly, we only worked on BPIC14 data and analysis. Future work is needed to analyse another dataset to prove whether the presented queries also work for other dataset flawlessly or changes are needed.

Furthermore, we created two different type of Visualizations. Future work can work on implementing more visualizations and gain more insights.

Finally, our analysis were implemented manually. An automatization is necessary to reduce the human involvement. Future work can work on to search and translate everything among the presented solutions into an automatized tool.

Bibliography

[1] 10th international workshop on business process intelligence 2014. https://www.win.tue.nl/bpi/doku.php?id=2014:challenge. Accessed on 2020-10-08.

[2] Neo4j. https://neo4j.com/. Accessed on 2020-10-08.

[3] Quick reference bpi challenge 2014. https://www.win.tue.nl/bpi/lib/exe/fetch.php?media=2014:quick_reference_bpi_challenge_2014.pdf. Accessed on 2020-10-08.

[4] M. Blake, R. Pradeep, and L. Lewis. Designing an evaluation framework for it service management. *Elsevier*, 47:219–225, 2010.

[5] R. Dijkman, J. Gao, and A. e. a. Syamsiyah. Enabling efficient process mining on large data sets: realizing an in-database process mining operator. *Distrib Parallel Databases*, 38:227–253, 2020.

[6] S. Esser. A schema framework for graph event data. Master's thesis, Technisce Universiteit Eindhoven, Netherlands, 2020.

[7] S. Esser and D. Fahland. Multi-dimensional event data in graph databases, 2020.

[8] J. Hanser. Analysing itil data to provide fact based predictive model of workload to service desk from the software change process. https://www.win.tue.nl/bpi/lib/exe/fetch.php?media=2014:bpic2014_submission_2.pdf.

[9] R. Ian, W. Jim, and E. Emil. *Graph databases*. O'Reilly Media, Inc., 1st edition, 2013.

[10] J. Leander. Automated translation of event data from relational to graph databases. Master's thesis, Technisce Universiteit Eindhoven, Netherlands, 2020.

[11] S. M. H. Mahmud, M. A. Hossin, H. Jahan, S. R. H. Noori, and T. Bhuiyan. Csv-annotate: Generate annotated tables from csv file. In *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 71–75, 2018.

[12] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comput.*, 24:1235–1258, 1989.

[13] M. Romero and M. A. Rodriguez. A graph-oriented model and query language for events, 2007.

[14] E. Stefan and F. Dirk. Storing and querying multi dimensional process event logs using graph databases. *Springer International Publishing*, pages 632–644, 2019.

[15] W. van der Aalst et al. Process mining manifesto. In *Business Process Management Workshops*, pages 169–194, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

List of Figures

List of Tables

Appendix A

ProM

First click "import" button at the top right corner and then choose an import plugin. It will offer automatically "CSV File(XES Conversion with Log Package)" option, choose this option and click "OK".

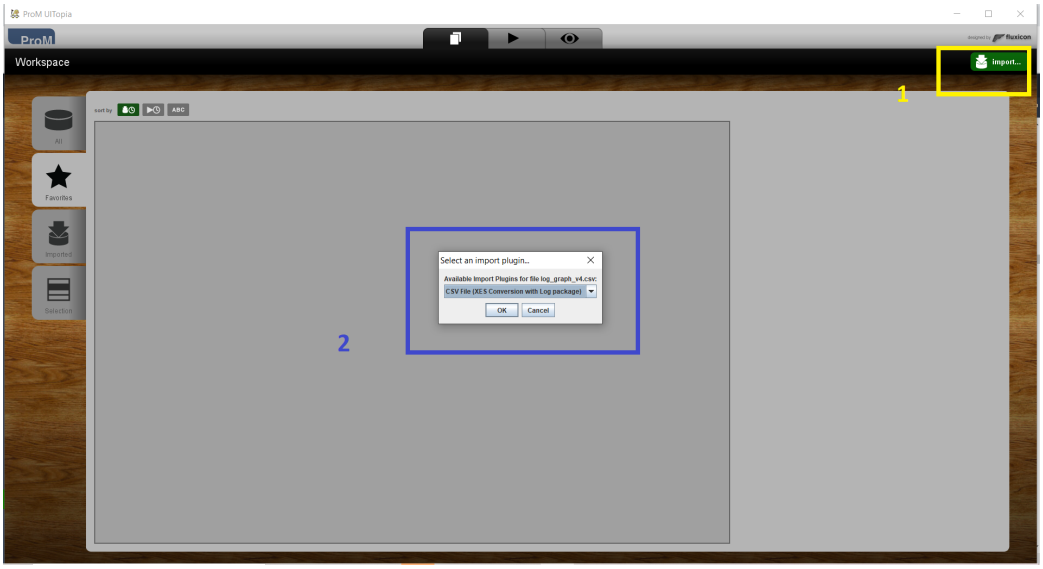


Figure A.1: Uploading the CSV file into ProM. First Import then Choose the import plugin "CSV File(XES Conversion with Log Package)".

It can take a bit time for ProM to upload the file. After uploading, you can click the "play" button and choose an algorithm. It will automatically offer "Convert CSV to XES", choose that, and click "play" button. See Fig.A.2 and A.3.

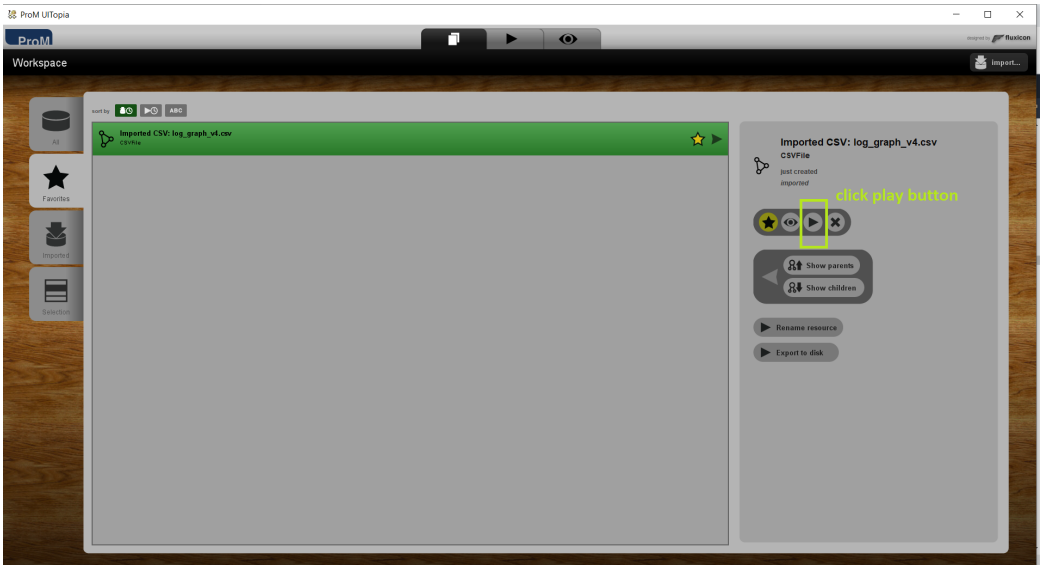


Figure A.2: Choose the file imported and click Play button.

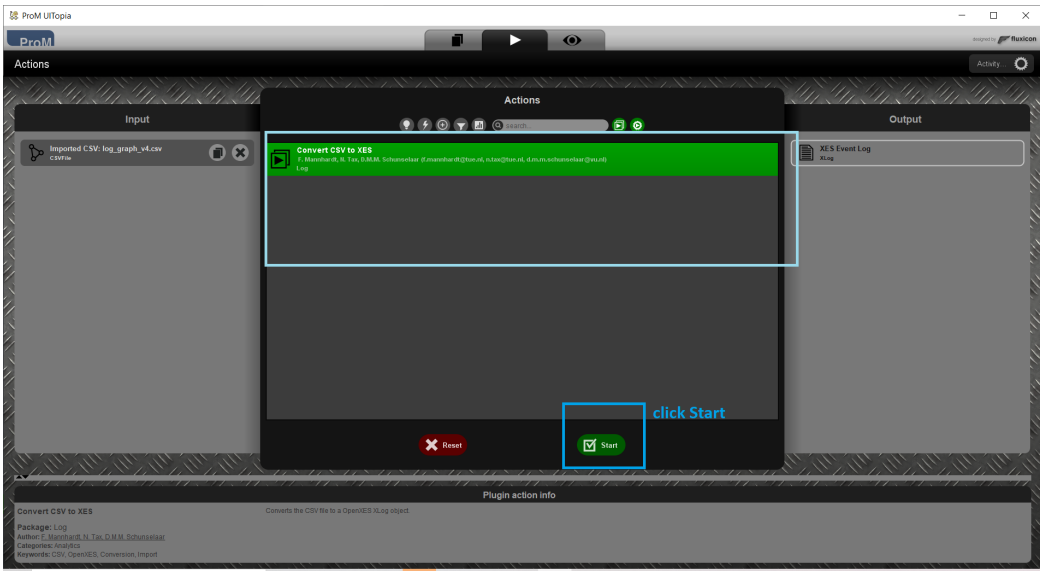


Figure A.3: Choose "Convert CSV to XES" and click "play" button.

Until coming the page in Fig. A.4, click "Next" for everything. Then choose "CI_ID" as case and "ent" as activity. Then click "Next" for the further steps. Preparing the file can take a bit time.

Figure A.4: choose "CI_ID" as case and "ent" as activity.

Then choose the 'Dotted Chart' option from the 'Log Visualizer' at top right corner. You can choose 'E: max_date' for X-axis and 'T:concept:name' for Y-axis and you will have the following result.

Figure A.5: Steps to create the Dotted Chart.

Appendix B

Impact Analysis For Interactions

```
ent_name=[]
ent_ID=[]#entity IDs
ch_list=[] #list of Changes
list_ent=[] #list of Entities
list_ent_ID=[] #entity IDs
prev=pd.DataFrame(columns=['CI','Change', 'prev_int'])
post=pd.DataFrame(columns=['CI','Change', 'post_int'])
file=pd.read_csv(CI_ID+".csv") #reads the CI file
list_ent=file.ent.tolist()
list_ent_ID=file.en_ID.tolist()
if 'Interaction' in list_ent and 'Change' in list_ent: #checks
   Interactions and Changes
    index= list_ent.index('Interaction') #finds the first
       Interaction
    for row in range(index, len(list_ent)):#works on the list
       after first Interaction
            ent_name.append(list_ent[row])
            ent_ID.append(list_ent_ID[row])
    for i in range (len(ent_name)):
            if ent_name[i] == "Change":
                 ch_list.append(ent_ID[i])

    first=0
    second=1
    for i in range (len(ch_list)-1):
```

```
if first==0: #first Change
        index_first=ent_ID.index(ch_list[first])
           #finds the index Changes in entity list
        index_sec=ent_ID.index(ch_list[second])
           #finds the index Changes in entity list
        cnt_int=index_sec-index_first # Interactions
           between two Changes
        prev=prev.append({'CI':CI_ID,'Change':ch_list[first],
        'prev_int':index_first},ignore_index=True)#first
           change previous Interactions
        post=post.append({'CI':CI_ID,'Change':ch_list[first],

           'post_int':cnt_int-1},ignore_index=True)
           #first Change post Interaction
        prev=prev.append({'CI':CI_ID,'Change':ch_list[second],
           'prev_int':cnt_int-1},ignore_index=True)
           #first Change post Interaction = second
           Change previous Interaction
        first=second
        second=second+1

elif second<len(ch_list)-1:

        index_first=ent_ID.index(ch_list[first])
        index_sec=ent_ID.index(ch_list[second])
        cnt_int=index_sec-index_first
        if cnt_int > 1: #if there is any Interaction
           between two Changes
            post=post.append({'CI':CI_ID,'Change':ch_list[first],
               'post_int':cnt_int-1},ignore_index=True)
            prev=prev.append({'CI':CI_ID,'Change':ch_list[second],
               'prev_int':cnt_int-1},ignore_index=True)

        else:
            post=post.append({'CI':CI_ID,'Change':ch_list[first],
               'post_int':0},ignore_index=True)
            prev=prev.append({'CI':CI_ID,'Change':ch_list[second],
               'prev_int':0},ignore_index=True)

        first=second
```

```
                second=second+1
            else: # last Change

                index_first=ent_ID.index(ch_list[first])
                index_sec=ent_ID.index(ch_list[second])
                cnt_int=index_sec-index_first
                post=post.append({'CI':CI_ID,'Change':ch_list[first],
                    'post_int':cnt_int-1},ignore_index=True)
                prev=prev.append({'CI':CI_ID,'Change':ch_list[second],
                    'prev_int':cnt_int-1},ignore_index=True)
                post=post.append({'CI':CI_ID,'Change':ch_list[second],
                    'post_int':len(ent_ID)-index_sec-1},ignore_index=True)
                    # Substracts the Change index in
                    entities list. The last Interactions
                    will be written.


    else:


            result=prev.merge(post,on =['CI', 'Change'],
                how='right')
            print(result)
            result.to_csv("results.csv", mode='a',
                header=False) #all CI's is kept in results.csv .

else:
    print(" There is no interaction and change") # Skip CI if
        there is no Interaction or Change or both.
```

Appendix C

Impact Analysis For Incidents

```
ent_name=[]
ent_ID=[]#entity IDs
ch_list=[] #list of Changes
list_ent=[] #list of Entities
list_ent_ID=[] #entity IDs
prev=pd.DataFrame(columns=['CI','Change', 'prev_inc'])
post=pd.DataFrame(columns=['CI','Change', 'post_inc'])
file=pd.read_csv(CI_ID+".csv") #reads the CI file
list_ent=file.ent.tolist()
list_ent_ID=file.en_ID.tolist()
if 'Incident' in list_ent and 'Change' in list_ent: #checks
   Incidents and Changes
    index= list_ent.index('Incident') #finds the first
       Incident.
    for row in range(index, len(list_ent)): #works on the list
       after first Incident
            ent_name.append(list_ent[row])
            ent_ID.append(list_ent_ID[row])
    for i in range (len(ent_name)):
            if ent_name[i] == "Change":
                 ch_list.append(ent_ID[i])

    first=0
    second=1
    for i in range (len(ch_list)-1):

            if first==0: #first Change
```

```
            index_first=ent_ID.index(ch_list[first])
                #finds the index Changes in entity list
            index_sec=ent_ID.index(ch_list[second])
                #finds the index Changes in entity list
            cnt_inc=index_sec-index_first # Incidents
                between two Changes
            prev=prev.append({'CI':CI_ID,'Change':ch_list[first],
                'prev_inc':index_first},ignore_index=True)#first
                change previous Incidents
            post=post.append({'CI':CI_ID,'Change':ch_list[first],
            'post_inc':cnt_inc-1},ignore_index=True)
                #first Change post Incident
            prev=prev.append({'CI':CI_ID,'Change':ch_list[second],
                'prev_inc':cnt_inc-1},ignore_index=True)
                #first Change post Incident = second
                Change previous Incident
            first=second
            second=second+1

        elif second<len(ch_list)-1:

            index_first=ent_ID.index(ch_list[first])
            index_sec=ent_ID.index(ch_list[second])
            cnt_inc=index_sec-index_first
            if cnt_inc > 1: #if there is any Incident
                between two Changes
                    post=post.append({'CI':CI_ID,'Change':ch_list[first],
                        'post_inc':cnt_inc-1},ignore_index=True)
                    prev=prev.append({'CI':CI_ID,'Change':ch_list[second],
                        'prev_inc':cnt_inc-1},ignore_index=True)

            else:
                    post=post.append({'CI':CI_ID,'Change':ch_list[first],
                        'post_inc':0},ignore_index=True)
                    prev=prev.append({'CI':CI_ID,'Change':ch_list[second],
                        'prev_inc':0},ignore_index=True)

            first=second
            second=second+1
        else: # last Change
```

```
                    index_first=ent_ID.index(ch_list[first])
                    index_sec=ent_ID.index(ch_list[second])
                    cnt_inc=index_sec-index_first
                    post=post.append({'CI':CI_ID,'Change':ch_list[first],
                        'post_inc':cnt_inc-1},ignore_index=True)
                    prev=prev.append({'CI':CI_ID,'Change':ch_list[second],
                        'prev_inc':cnt_inc-1},ignore_index=True)
                    post=post.append({'CI':CI_ID,'Change':ch_list[second],
                        'post_inc':len(ent_ID)-index_sec-1},ignore_index=True)
                        # Substracts the Change index in
                        entities list. The last Incidents will
                        be written.


    else:
            result=prev.merge(post,on =['CI', 'Change'],
                how='right')
            print(result)
            result.to_csv("results.csv", mode='a',
                header=False) #all CI's is kept in results.csv .
else:
    print(" There is no Incident and change") # Skip CI if
        there is no Incident or Change or both.
```

Appendix D

Visualization of Paths - Interactions and Change

Figure D.1: Changes effecting more than one Configuration Item.

Appendix E

Visualization of Paths - Incidents and Change

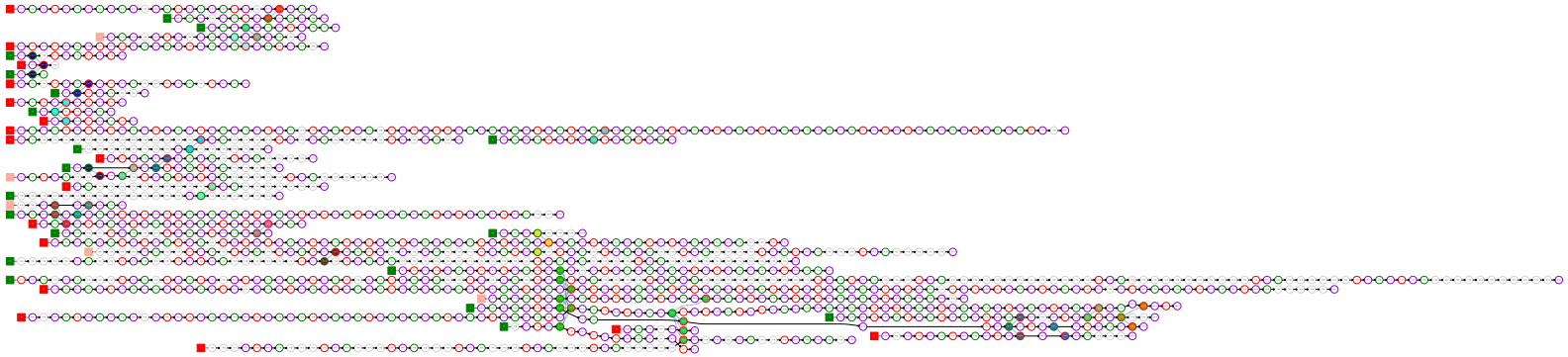


Figure E.1: Changes effecting more than one Configuration Item-Incidents.

Appendix F

Graph Components of LSR000160 in Esser's Schema

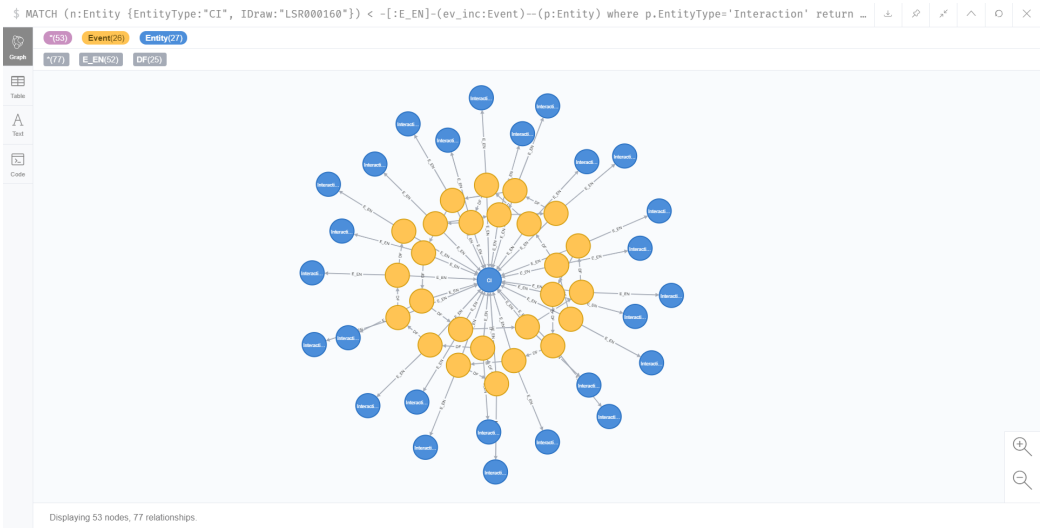


Figure F.1: Graph Component 1 of LSR000160.

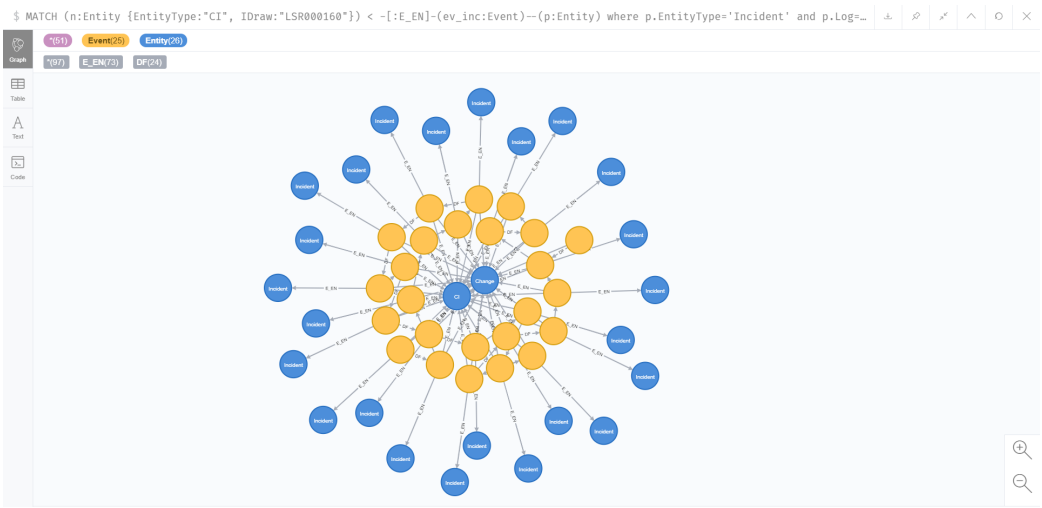


Figure F.2: Graph Component 2. Only has Incidents.

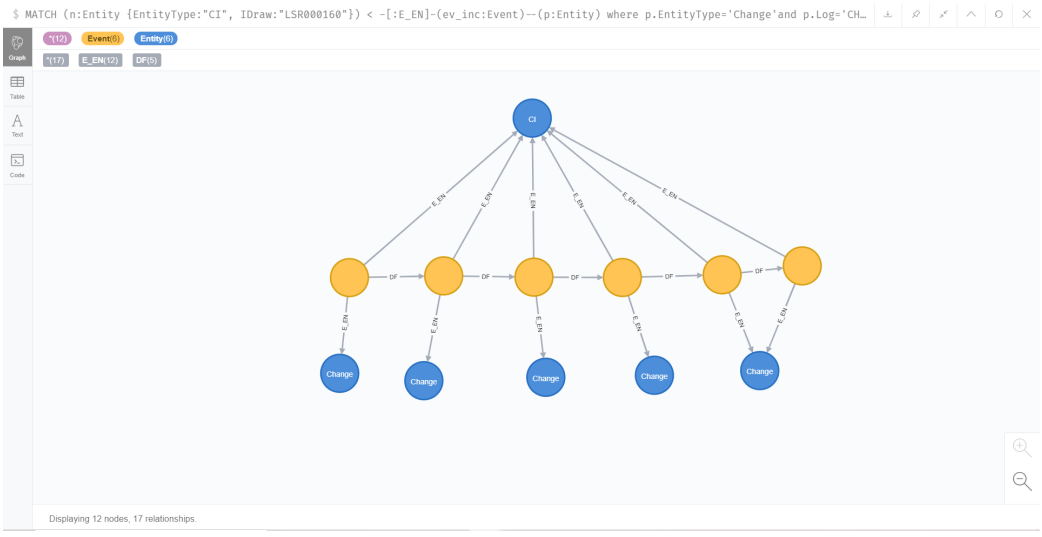


Figure F.3: Graph Component 3 with Change entities.

Appendix G

Analysis Schema Example Query

Following query shows how to receive a path for a particular Configuration Item.

```
match(n:CI{id:'5613'})--(en:Entity{CI_ID:'5613'})--
(en2:Entity{CI_ID:'5613'})
return en,en2
```

The result of the query is in Fig.G.1.

Figure G.1: Result of Query 19. CI:5613 and its path.

Appendix H

Result of First Option in Impact Analysis

We ignore other changes on the path, therefore, we could not decide which Interactions/Incidents occurred because of which changes. Fig. H.1 shows an example of a series of Changes and Interactions/Incidents.

Figure H.1: Calculating impact of Changes-First option

Blue nodes are Incidents/Interactions whereas yellow ones are representing Changes. As the path does not start with an Interaction/Incident we ignore CHG1 to obtain the first Interaction/Incident. With the first way the result would be as follows:

The Interactions at the beginning may not be related with CHG5 but definitely related with CHG2. Therefore, we do not need to include that information for CHG5.

Table H.1: Result of First option

Entities	**previous Incidents/Interactions**	**post Incidents/Interactions**
CHG2	3	7
CHG3	5	5
CHG4	7	3
CHG 5	10	0