Utrecht University
Department of Information and Computing Sciences

Master Thesis

InteractionCity:

Re-discovering system context knowledge using interaction data

Author:

B.N. Janssen, BSc
b.n.janssen@students.uu.nl
5661145

Supervisors:

Dr. ir. J.M.E.M. van der Werf
J.M.E.M.vanderWerf@uu.nl

Prof. dr. ir. A.C. Telea
a.c.telea@uu.nl

February, 2023

Utrecht University

Abstract

Context knowledge of a system can be not available at all, when it is available its often undocumented and has a low bus factor. In order to recreate such context knowledge, we propose a new visualization technique InteractionCity. This work uses the techniques proposed in Visual Analytics as a basis for the knowledge obtaining process and builds on the work of ArchitectureCity [24].

The technique only uses execution data to visualize both the static and dynamic parts of the system. The execution data used is based on a new concept; interactions. This aims to both simplify and improve the processing of the data, contrary to existing event-based models. An interaction connects two objects, denoted through their FQN, and provides additional information on said interaction.

Finally, the created design and implementation for InteractionCity is evaluated through a case study. We conclude the research project with various possibilities for future work, both interaction-based and InteractionCity-based.

Keywords: Process mining, Software Architecture, Interactions, Visualization

Acknowledgements

After many years, my academic career seems to be coming to an end. In a bit more than the past year, I have worked many hours to finalize this research project into what it is now. However, I would not have been able to create this master thesis without the support of the following people, whom I thank greatly.

Firstly, my first supervisor, Jan Martijn van der Werf, who guided me during the entire process and help shape the results. My second supervisor, Alex Telea, who made time to provide feedback, even with a tight schedule. Furthermore, I would like to thank Camille van Dijk, my girlfriend, who supported me, when working on this was though and with whom I got through times of Covid. Lastly, I also want to thank everyone from the shared study space (dubbed "Høk") that provided a motivational work environment and interesting discussions, with a special mention of Willem Hulst.

Brian Janssen
February 13, 2023

Contents

Context

Chapter 1

Introduction

Each software system has an architecture and runtime behaviour. The design of a system states how the structures of the software should relate to one another and their interactions [25], in theory. The actual behaviour of the system and the component interactions, such as defects in the order messages are sent, depend on the context, such as specific hardware, within which the software executes [27]. Hence, the actual behaviour cannot be contained within the design of the system, instead need to be collected during runtime. In the ideal world, the actual usage is a subset of the designed usage and is fully understood and well documented, such that the architects can use this info to make further design decisions.

In many real world systems, the development of these systems is performed in small and quick iterations to keep up with the high time to market requirements [8]. Additionally, some of these systems are split amongst multiple distributed components, which might be developed by different teams [1, 25]. These aspects result in a smaller focus on the maintenance needed which keeps the documentation and knowledge of the system up to date and relevant [1]. This becomes even worse when the design and context knowledge has a low bus-factor [4] and the developers owning that knowledge leave.

Existing techniques to reverse engineer, i.e. recreate, such design documentation and knowledge, often require specific inputs or do not focus on both the design and behaviour of the system. These inputs might not necessarily be available, e.g. as these are lost or proprietary. When some, possibly limited, contextual knowledge is available through domain experts, recreating this input might not be feasible [12], especially when it's "feeling" about the system.

In the case of design recreation techniques, this can be seen where certain design patterns are misinterpreted, such as observer patterns, whereas in the case of behaviour recreation techniques it's hard to know what is actual unexpected behaviour [32]. Hence, humans should be able to guide and interpret the recreation process, to alleviate these problems.

To obtain insights of the system, visualizations techniques can be used [28]. Many various visualization techniques exist [7], each with various specializations, and can be used to visualize

both design and behaviour. The visualization can then be presented to the user, who can interpret it within context and obtain knowledge of said system. However, the number of techniques which focus on using only execution data to visualize both design and behaviour to obtain a basic context knowledge is slim. As such, the design objective of this research project is:

Improve the architectural and context knowledge *by* creating a new visualization technique *that* uses the system's execution data *in order to* improve the understanding of said system.

Chapter 2 Research Approach

As expressed in the problem statement, this research will focus on creating a new visualization technique. In this chapter, we outline the research, and present the questions and methods to systematically arrive at a solution for our problem statement.

2.1 Research Questions

To solve the problem stated in the previous chapter, we focus on answering the following main research question:

RQ: What are techniques to support the architect in understanding the behaviour of a software system?

This research question can be decomposed in the multiple sub-questions. We start with sub-questions regarding the context of the problem and possible solutions. This gives us the first two sub-questions:

SQ1: How can we capture and represent dynamic information about the usage of software systems efficiently?

SQ2: What is the current state of the art in visualizing static and dynamic information for software architecture?

A visualization technique consists of a collection of different key components, each with a different function. Creating such component for each element and combining these into a single view results in the final visualization of the system. This leads to the question:

SQ3: What visualization techniques can be developed to improve the understanding of a software system's behaviour?

Phase	SQ	Method	Chapters
Problem Investigation	1, 2	Literature Study	3, 4, 5
Treatment Design	3	Agile	6
Treatment Validation	3	Agile	8
Treatment Implementation	3	Agile	7
Implementation Evaluation	4	Agile, Case Study	8

Table 2.1: Overview of this research project's Engineering Cycle phases and their accompanying research questions and chapters.

Part of this question is the implementation of the envisioned technique. This implementation needs to be evaluated.

SQ4: How is the developed visualization technique perceived and compare against existing techniques?

By answering all the sub-questions, the answer to the main research question can be given.

2.2 Research Method and Document Structure

By creating a new technique, this research project can be characterized as Design Science [34]. In this research, we follow the Engineering Cycle as a rational problem-solving process, which consists of four phases: *Problem Investigation*, *Treatment Design*, *Treatment Validation*, and *Treatment implementation*. Once the *Problem Investigation* has been performed, this phase is transformed into the *Implementation Evaluation* phase.

During this research project, the Engineering Cycle will be performed a various number of times. However, only the final combined results will be presented, rather than listing the results of each individual iteration step. When major changes are made in one of the iterations, these will be highlighted in their respective chapters.

Within the *Problem Investigation* a literature study will be performed. This is a semi-structured snowballing literature study, which started with the work of [23, 24]. The *Implementation Evaluation* phase will consist of a Case Study, in which the, then, proposed visualization technique is applied on a real-world dataset.

The final mapping between the Design Science phases of this research project and the structure of this document is shown in Table 2.1. Additionally, each of the phases can be linked to the proposed research questions. Chapters 1 and 9 are not part of the Engineering Cycle, instead providing the introduction and conclusion to this research project rather than the technique itself.

Additionally, this document is structured in four segments; Context, Background, InteractionCity, and Evaluation. Each of these parts groups related concepts together and loosely relates to the different phases. Context introduces the research problem evaluated in this study. Background introduces various topics required to understand the context. InteractionCity describes the design and implementation of the implemented visualization technique. Lastly, Evaluation discusses the results of the created visualization technique and the results of this study.


Part
II
Background

Chapter 3
Visual Analytics in Software Engineering

In this chapter, we focus on the basics of data visualization, and how it is used in Software Engineering. There are various methods to visualize different types of data. Visual Analytics is such a generic framework to transform data into knowledge. Its goal is to guide the information processing steps in order to help the evaluation and improvement of the techniques. This in turn will improve the obtained knowledge and following decisions through visualization. In this thesis, we adopt the definition of [16]:

Definition 3.1, Visual Analytics [16]:
Visual Analytics combines automated analysis techniques with interactive visualizations for an effective understanding, reasoning and decision-making on the basis of very large and complex data sets.

Although the definition of Visual Analytics can be used for any visualization and knowledge obtaining task, this research project focuses on Software Engineering. The concept of applying visualization techniques on Software is not a new research area, and as such we will look at previous works in Section 3.3.

In this research project, the Visual Analytics framework will be used as a basis to build our visualization technique on and to reason about it. It will be a basis for the end user to perform the transformation of data to knowledge and describe the conceptual framework to create the visualization technique.

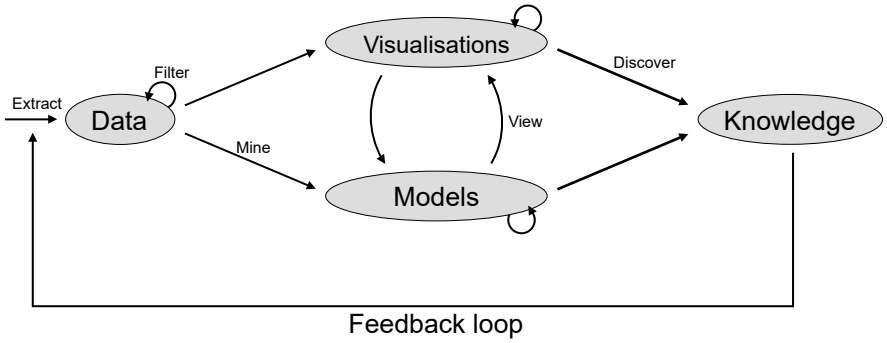


Figure 3.1: Visual Data-Exploration as proposed by [16, Figure 1.]. It shows the tight integration of visual and automatic data analysis methods with database technology for a scalable interactive decision support.

3.1 Visual Data Exploration

A visual overview of the Visual Analytics framework is shown in Figure 3.1. It shows four aspects: *Data*, *Models*, *Visualizations*, and *Knowledge*. Each of these aspects is connected to (some) of the other aspects through the means of an directed arrow. A directed connection indicates what aspects provide input to other aspects. By following the directed connections we can continue obtaining knowledge.

This knowledge is the final output, with which the user can reason about the system and make decisions accordingly. This new knowledge can in turn be re-used to better refine the entire process, hence there is a feedback loop resulting from the knowledge. The framework is in an never-ending loop, since obtaining knowledge and refining the system will be, given that the system does not cease to exist, will not be redundant.

Knowledge is obtained through the input of the *Visualizations* shown to the user. The number of these *Visualizations* may vary, depending on the wants and needs of the user. A *Visualization* has two additional outgoing connections, one which loops with itself and one with *Models*. The looping connection denotes that a *Visualization* itself can provide input for new *Visualizations*, where the new visualization is an improved version of the initial one without having to modify the other input values.

The *Models* are the underlying data of the *Visualization*, hence a connection from *Models* to *Visualizations*. This *Model*'s data is a transformed version of its input *Data*, providing insights that otherwise could not be obtained. These transformations can both be simple, e.g. a sum or sorting, or complex, e.g. Principal Component Analysis, depending on the needs of the user. The *Model* is also an input for the *Knowledge* aspect, since the *Visualizations* might not provide all wanted information, and the actual data values are needed as well. Similar to *Visualizations*, *Models* has a loop towards itself, which allows it to modify it's raw data transformations, without changing the input *Data*. The connection from *Visualizations* to *Models* allows the *Visualization* to modify the available data mined in the *Model*.

The last aspect is the *Data*, which is the raw data obtained from the system (i.e. the logs). Often, this data needs to be cleaned, files need to be joined, and anonymized. This process might need several passes before a sufficient quality is obtained, hence there is a looping connection. The data is not only input for the *Models*, but also for the *Visualizations*. Having such direct connection allows the visualization to show some raw data values, rather than only the *Model*'s data.

All connections between the aspects are visualized the same, however these connections are used in varying degree. The default mode of operation in the Visual Analytics framework starts with the *Data* aspect. This *Data* is processed to obtain a *Model*, which in turn can be used for a *Visualization*. Lastly the *Knowledge* is obtained by the user. The other connections only are used in specific instances.

3.2 Requirements for Visual Analytics Techniques

In order to apply Visual Analytics, the system needs to fulfil a set of criteria. These requirements are listed and described by [16] throughout their paper. They do not affect the visual representation of the system directly, but rather influence the system as a whole on a structural level. These requirements can be split in both functional and non-functional requirements. The functional requirements denote what the technique should be able to perform, such as displaying specific attributes. Non-functional requirements can be seen as how the technique performs the functional requirements and how the user perceives the technique. Summarized, these requirements are:

Interactivity (Non-Functional)
The system needs to be interactive. These interactions should result in immediate feedback, i.e. work in real-time. By using the interactions the user should be able to change the current visualization or modify the currently visualized data. These interactions and the additional data obtained through them, allow the user to better comprehend the system. In turn, a better understanding allows the user to better obtain possible knowledge from the system.

Intuitivity (Non-Functional)
The User Interface (UI) needs to be intuitive and require only minimal input. The goal is to minimize the barrier between what the human wants to accomplish and what the computer's understanding is of what want. The feedback given by the user to the system should in turn also be taken as intelligently as possible. This allows the user to fully focus on the task at hand, obtaining knowledge, rather than be distracted by the complex UI.

Understandability (Non-Functional)
The mapping from the input data to the spatial dimensions with which the data is visualized needs to be known to the user. By having a clear and understandable mapping from the data to the spatial representation, decreases the possibility of misinterpretations and increases the understanding of the visualization shown to the user.

Scalability (Non-Functional)
The system needs to be able to scale with the size and dimensionality of the input data, as the system needs to be interactive and the amount of data can be very large.

Level of Detail (Functional)
The visualization needs to be able to visualize the data at several levels of detail, as the data can contain knowledge of all levels. The appropriate representation should be chosen by the user, depending on the knowledge the user wants to obtain. The visualization should start with a global overview of the system and become more detailed if required, as stated by [16]: "Analyse first, Show the Important, Zoom, filter and analyse further, Details on demand".

Data Quality (Functional)
The quality of the data and confidence of the algorithm used needs to be appropriately

represented in the visualization. By informing the use of these aspects, possible misleading analysis results can be avoided.

Asynchronous Infrastructure (Functional)
The system needs to manage the different algorithms and their data. The algorithms might require large amounts of memory and computational power, hence the system needs to be built on a novel infrastructure which can handle the large volume of data and asynchronously manage the processes.

3.3 Visual Analytics for Software Engineering

Using Visual Analytics for Software Engineering has been researched in other works. The amount of interactions possible with these visualization techniques is, however, often limited. A closely related field is the visualization of hierarchies, aside from Software Engineering, hence most software visualizations share features with this field.

The first of such visualizations is a rooted-tree [28] (Figure 3.2a), where the tree-like structure is used to show the hierarchical dependencies between elements. The elements of which the software consists are scaled by the actual size of the hierarchies and files contained within this group. Additional colouring is used to denote different abstraction levels in this hierarchy, where the edges are coloured depending on the two connected elements. Notably the colours of the edges are interpolated to transition smoothly, creating a visualization with less-sharp transitions [28]. Using such tree-visualization, with distinct horizontal layers, allows for a quick assessment of various aspect of the system, such as the size, depth, and number of elements within each component. These aspects in turn give an indication of the size and complexity of the system.

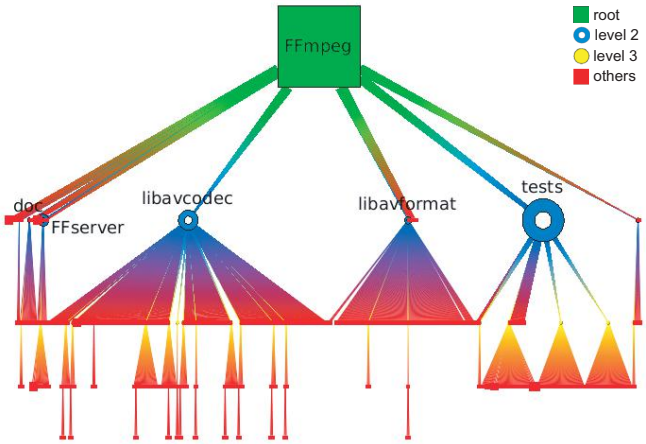
An alternative representation of the same file hierarchy is shown in Figure 3.2b. This visualization uses squares which are sized appropriately to the data it represents. Hence, treemaps are most useful when size, e.g. storage required, is the main aspect being visualized. These squares in turn are positioned in a squarified layout and shaded using cushion rendering. Squarified treemaps, compared to regular treemaps, result in less tall and elongated rectangles [2].

An alternative visualization, with a focus on interactions between elements rather than the size of the elements, are the call graphs shown in Figure 3.2c [28]. Notably the hierarchy of the elements is still visible within the outer rings of the graphs. The edges within the two call graphs display the interactions between the two connected elements. These edges are coloured in order to further visualize additional information, such as direction and type of call. Furthermore, graph `a` (Figure 3.2c) shows edge bundling in order to reduce the visual clutter, which is present in graph `b`.

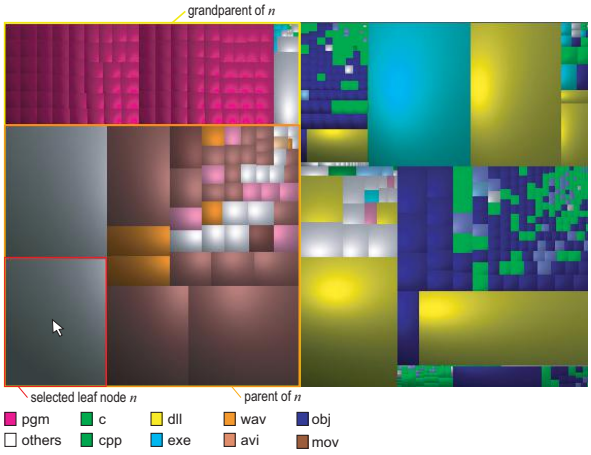
Finally, the CodeCity visualization [33] shown in 3.2d. This visualization focuses on the static aspect of software; code. Each class in the software results in a building, which is sized depending on various aspects of the class. In the figure shown the width of each building is scaled by the number of attributes, whilst the height is scaled by the number of functions. The classes are grouped by the package or namespace they belong to. Since the visualization only shows the static code, without either statically analysed or runtime interactions, it provides a similar view as the squarified treemap, Figure 3.2b, but gives an additional height dimension.

3.4 Summary

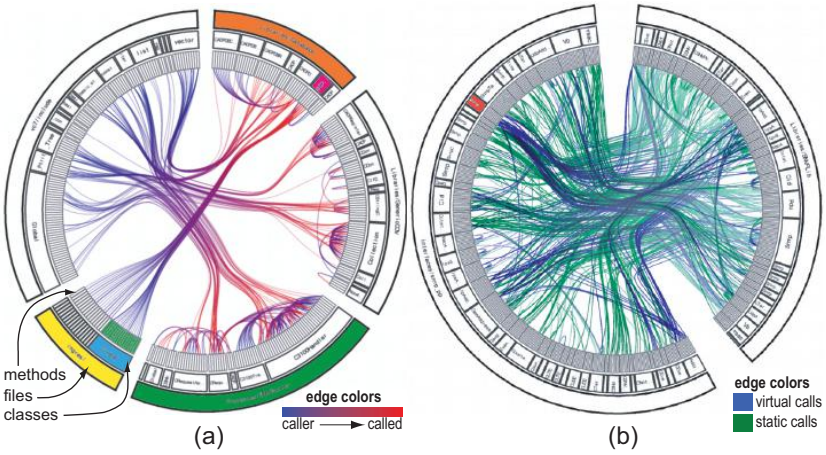
In this chapter, we have introduced and explained Visual Analytics, consisting of four different aspects. Visual Analytics will form the basis of this research, where the steps of the framework



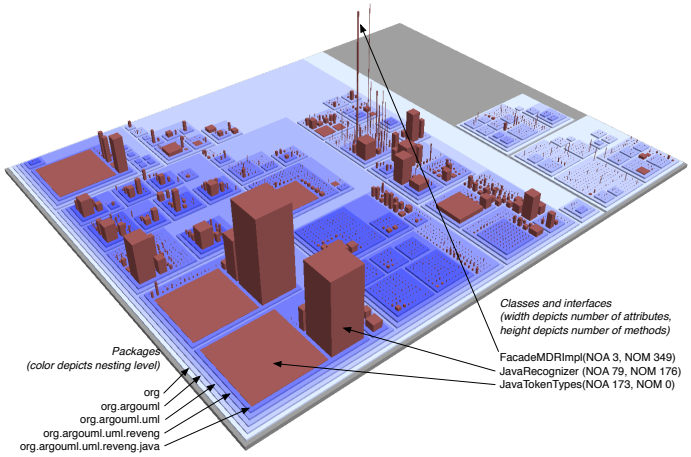
(a) File hierarchy of the FFmpeg software visualized using a rooted tree, by [28, Figure 11.5].



(b) Same software as Figure 3.2a rendered with a squarified layout and shaded cushion rendering, by [28, Figure 11.5].



(c) Call graphs of two programs visualized together with the programs' hierarchical layering, by [28, Figure 11.16].



(d) CodeCity overview for ArgoUML v.0.24, by [33, Figure 1].

Figure 3.2: Various Software Visualization techniques.

will be performed by the user in order to derive the knowledge from our visualization technique. The four aspects of Visual Analytics can be linked to our research questions.

RQ1 looks at the ways to capture and represent the *Data* aspect. In RQ2 the *Visualizations* aspect is answered. A combination and the interaction between *Models* and *Visualizations* will be looked at in RQ3. Lastly, the value of the obtained *Knowledge* is answered in RQ4.

Additionally, we have seen how visualization techniques have been used in prior research to help the developers obtain new knowledge. All of these techniques use some form of hierarchy visualization, but do not have a fixed metaphor in doing so. In these hierarchies, the focus lies on the structural dependencies between software elements, rather than the runtime behaviour that these elements have. Colours are also an important aspect of these visualizations, although are never used as the main focus of the visualization, instead the size of and connections between elements provide the initial knowledge.

Chapter 4 Software Architecture

This chapter focusses on RQ2: "What is the current state of the art in visualizing static and dynamic information for software architecture?". To answer this question, we first introduce Software Architecture. Next, we discuss visualization techniques currently available in Software architecture.

4.1 Software Architecture

Previously stated in the design objective and research questions, this research project will involve a system's architecture. Every system has an architecture, whether or not it is documented and understood [25]. Following [1], Software Architecture is defined as:

Definition 4.1, Software Architecture [1]:
The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.

This set of structures consists of three different types:

Modules are structures with each a computational responsibility. Modules split the architecture into implementation units (e.g. a set of code) and are often assigned to a single development team.

Components-and-connectors are the design time services that the system is composed of and their interactions. These structures have runtime behaviour and interactions which makes them dynamic in nature.

Allocation are the relations that the system has with non-software structures. They describe the mapping from the system to its environment, such as its organization and development.

Additional to the Software Architecture definition, [25] defines key concepts: Stakeholders, Viewpoints, Views, and Perspectives. These concepts are the core throughout their book, and are defined as follows:

Stakeholders
are persons, groups, or entities with an interest in or concerns about the realization of the architecture. Not only are the end-users of the system the stakeholders, but this also includes the developers and owners of the system.

Views
are a representation of a certain single aspect of the system's architecture. This representation illustrates how the system operates regarding that aspect. These aspects are the aspects in which a stakeholder might be interested. All knowledge required for a single stakeholder can be described in a particular set of views.

Viewpoints
are a collection of patterns, templates, and conventions for a single view. Additionally, it describes the Stakeholders which are interested in that view and describes how the view can be created. Viewpoints can be used to structure the creation and description of the Software Architecture, based on the principle of separation of concerns. To create the full set of views for a single Stakeholder, multiple Viewpoints might be needed to be applied.

Perspectives
are a collection of activities, tactics, and guidelines, complementary to Viewpoints. Where Viewpoints focus on a certain aspect of the architecture, perspectives focus on a certain quality of the architecture. These qualities are related to multiple components of the system and their relations, and thus require multiple views.

In short; we can communicate about a system by describing it though a Software Architecture. The Software Architecture is represented using a set of Views, which are created by applying multiple Viewpoints and Perspectives on the system. The chosen Viewpoints and Perspectives depend on the knowledge needed about the system.

This allows us to achieve the, in Chapter 1 stated, desired goal of describing architectural and context knowledge. Hence, it will play a main part in this research project. How the actual Views are represented, such as textual or visual, depend on the Viewpoint and Software Architecture technique used. Since our design objective focuses on a visual representation, the focus will also lie on visual representations when exploring existing Software Architecture techniques.

4.2 Roles and Purposes

Software Architecture can play an important role for various aspects. A total of six different roles are defined by [11]. Additionally, in their review, [26] found ten categories of purposes of using visualization techniques in Software Architecture. These six roles and ten purposes contain large amounts of overlap and are combined into the following list:

Understanding
High-level abstractions simplify the users ability to comprehend the system. This allows the user to improve its understanding of both static and dynamic characteristics of the

system. Additionally, the architectural design decisions can also be represented through these visualizations.

Construction
The architecture provides a blueprint for development, indicating the components and interactions. This can be used outside the actual development of the system for both reengineering and reverse engineering of the software. During the actual development of the system Software Architecture can also improvements, such as: supporting model-driven development, check compatibilities and synchronization between design and implementation, and show components that can be reused at various levels.

Analysis
Architectural descriptions allow for various analysis. These analyses can provide insights into violations, flaws, and faults in the architecture design, such as requirements checking. Additionally, the analysis can be used to expose parts that can or should evolve.

Management
By having a good Software Architecture, searching, navigating, and exploring the architectural design of the system can be much improved. It also provides traceability between architecture entities and software artefacts. Both of these aspects allow for a better understanding on a management level, which typically leads to better requirements, strategies, and risk assessment.

Which role(s) the usage of Software Architecture fulfils depends on the needs of the user. The *Analysis* and *Management* roles require a Software Architecture, or a good understanding of it, to be known prior to the application of the technique. However, as stated in the design objective, current knowledge of the system and its architecture is lacking. Rather, the objective is to obtain this knowledge, which eliminates the *Construction* as relevant roles. The *Understanding* role is thus the role which will be the focus when using Software Architecture within this research project.

4.3 Software Architecture Reconstruction

Software Architecture by itself is only a descriptive tool. However, many systems lack such a proper architecture documentation [6], hence research has been conducted into reconstructing the Software Architecture of existing software: Software Architecture Reconstruction. In the work of [8, 18], Software Architecture Reconstruction is defined as:

Definition 4.2, Software Architecture Reconstruction [8]:
Software architecture reconstruction is a reverse engineering approach that aims at reconstructing viable architectural views of a software application.

Other terms for Software Architecture Reconstruction are: reverse architecting, or architecture extraction, mining, recovery or discovery [8]. The reconstruction itself is an iterative process and requires human knowledge to guide it and validate the results. The field of Software Architecture Reconstruction is structured among five axes by [8], as shown in Figure 4.1.

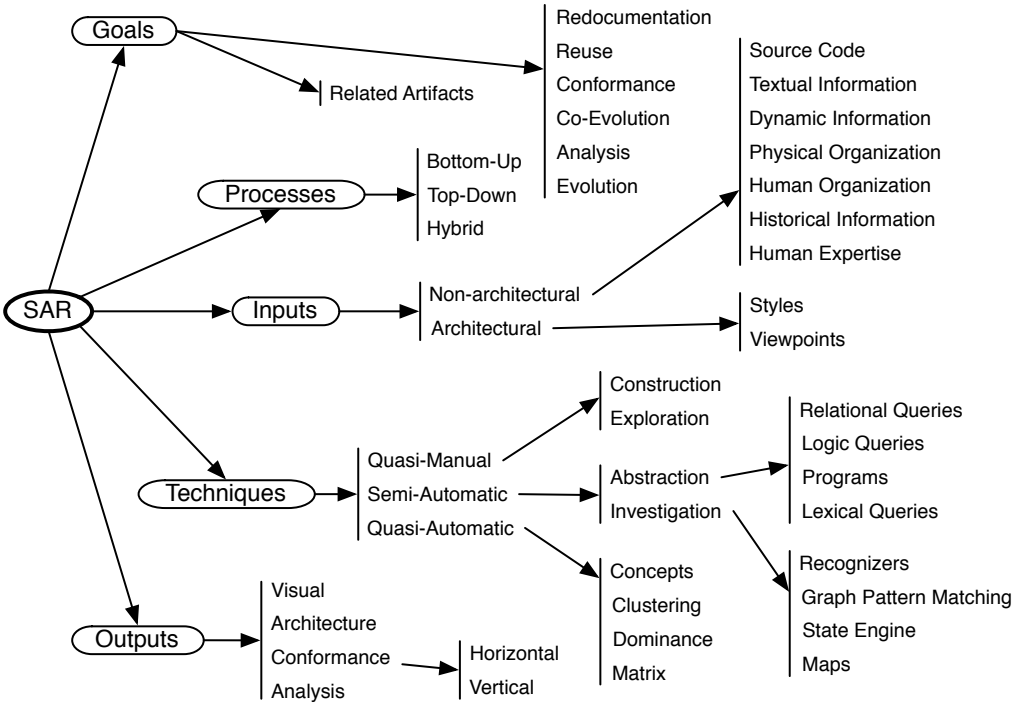


Figure 4.1: A process-oriented taxonomy for Software Architecture Reconstruction, as proposed by [8, Fig 2.].

Goals

The first axis are the goals, which are similar to the roles described previously in Section 4.2.

Processes

There are multiple approaches possible when reconstructing the architecture: Bottom-Up, Top-Down, and Hybrid. All approaches are iterative, where they start with a certain starting state and the event log. The starting state is transformed and the process is restarted. Once a desired abstraction level has been achieved, the process stops. Literature also refers to a Bottom-Up process as *recovery* and to a Top-Down process as *discovery*.

A Bottom-Up process uses low-level knowledge, such as source code, about the system as the initial state for the architecture. This knowledge is then transformed, resulting in a higher-level abstraction. In a Top-Down process the opposite is done; it starts with high-level abstracted knowledge, such as requirements and architectural styles. This high-level knowledge is compared to the actual event logs and refined where it does not match, resulting in a lower level abstraction. When using a Hybrid approach, both Bottom-Up and Top-Down are used. First, a Bottom-Up approach is taken, using the low-level knowledge to create an initial state. This initial state can then be pass to a more Top- Down approach, further refining this state.

In this research, data from the system consists of only the execution data, thus no low-level information, such as source code, is known. Hence, the Architecture Reconstruction method used can only be a Top-Down approach.

Inputs

The third axis is the Input, split in *architectural* and *non- architectural* inputs. The Input denotes the data available to the technique, including non-system related information. *Architectural* inputs are related to the Software Architecture as a concept, regardless of the actual system. This includes the Viewpoints, Perspectives, and architectural styles used by the stakeholders when discovering the system. The actual data of the system, such as source code, execution data, and possible expertise, are the *non-architectural* inputs. The *architectural* inputs, dynamic information, and human expertise will be the only inputs used in our visualization technique.

Techniques

The technique can be classified on its level of automation, which forms the fourth axis: technique.

Quasi-manual The reverse engineer manually identifies architectural elements using a tool to assist him to understand his findings.

Semi-automatic The reverse engineer manually instructs the tool how it can automatically discover refinements or recover abstractions.

Quasi-automatic The tool has the control and the reverse engineer only steers the iterative recovery process.

Each technique requires at least some human guidance, however the level of automation chosen does not depend on or sets requirements for the input data. Due to the lack of requirements with these two aspects, no elimination can be made and can be chosen depending on the context of our visualization technique. Additionally, the three classifications are not mutually exclusive, so techniques are able to choose a level of automation for each sub-part of the algorithm.

Outputs

The last axis is the output axis, consisting of various types:

Visual is shown to the user to get a better understanding of the architecture. This visual representation does not make explicit observations by itself, instead lets the user make the observations from the visualization.

Architecture often consists of various *views*, rather than an (interactive) visualizations. Instead, architecture output uses the same methods to present the information as Software Architecture Views, such as an Architecture Description Language.

Conformance requires an existing architecture with which the found architecture is compared. With this comparison, the actual implemented application its conformance to the prior known architecture can be determined.

Analysis The analysis obtains information about the architecture. This information can be used to qualify or refine a prior known architecture.

In this research project, the main goal of the designed technique is a visualization. Hence, when using a Software Architecture Reconstruction technique, one with a focus on the visual aspects should be chosen.

4.4 Requirements for Software Architecture Visualization Techniques

The definition of Software Architecture is indifferent whether the architecture for a system is a good one or a bad one and if it is a good one whether it is fit for its purpose [1]. Hence, requirements need to be set such that the made model fulfils its purpose. Derived from [1], these requirements are:

Reasoning (Non-Functional)
The Software Architecture must support reasoning about the system and the system its properties. A set of structures that does not allow for a stakeholder to reason about the system its architecture does not fulfil the purpose of obtaining knowledge.

Abstraction (Non-Functional)
The Architecture should not include elements which do not provide additional useful information when reasoning about the system, i.e. the Software Architecture provides an abstraction. The user cannot deal with the increased complexity all the time, when too much superfluous information is presented. This implies that information about a single structures which does not have any effect on other structures, is omitted.

Includes Behaviour (Functional)
The Software Architecture needs to document the behaviour of the software elements. The interactions presented in a Software Architecture form an important part, as they might influence other software elements or the system as a whole. However, not every behaviour needs to be documented, but should instead be tuned to the required abstraction level.

Additionally, [8] lists several requirements for Software Architecture Reconstruction.

Data Quality (Non-Functional)
Architecture Reconstruction needs to take the quality of the data into account.

Scalability (Non-Functional)
The technique should scale with the, possibly, very large datasets.

Top-Down (Functional)
Since only the system's execution data is available, a Top-Down reconstruction approach must be used.

Views (Functional)
Multiple architectural *viewpoints* should be supported in order to provide the user with multiple options during the discovery process.

Iterative and Parameterizable (Functional)
The Architecture Reconstruction process needs to be interactive, iterative, and parameterizable. This allows the user to further refine the architecture during the discovery process in incremental steps, and thus reduce the complexity.

4.5 Software Architecture Visualization Techniques

The purpose that Software Architecture can fill depends on the technique and implementation chosen. In this chapter, we will look at some existing techniques, and their technique, and the requirements that Software Architecture poses on such tool.

In the work of [23, 26], four different types of Software Architecture visualizations are listed:

Graph-Based Techniques use nodes and links to visualize relations between elements.

Notation-Based Techniques use modelling techniques consisting of: unified modelling language (UML), systems modelling language (SysML), and specific notation-based visualization.

Matrix-Based Techniques use the matrix-format to display data when its graph is large or dense.

Metaphor-Based Techniques use familiar physical world contexts to visualize both elements and their relations.

A Notation-Based Technique, such as UML, can be interpreted as a graph with additional visualization constraints and various assigned attributes. Hence, the Notation-Based Techniques will be considered a subset of the Graph-Based Techniques. In this study, these three types will be adopted as categories of Visualization Techniques.

On top of this classification, [23] cites three design choices for arranging networks: node-link diagram, adjacency matrix, and enclosure. The node-link digram and adjacency matrix are similar to the Graph-Based Technique and Matrix-Based Technique respectively. The enclosure does not have obvious overlap with the three classifications. Here relationships are shown using containers in which the elements are grouped and can provide hierarchical ordering within these relations, such as in a treemap.

Several varying visualization techniques are listed by [23, 26]. Notably, [23] lists the characteristics/requirements that should be taken into account of visualizations techniques, being: multiplicity of view, dimensionality, medium, interactivity, implementation, and data representation.

Further literature proposes alternative visualization techniques, some of which do not entirely fit the prior mentioned classification. The flow-model technique proposed by [14], is such method. This method uses both static and dynamic inputs, which is merged into a single input and visualized. Multiple visualizations are proposed, one of which is shown in Figure 4.2a, thus multiple classes are applicable. Notable, it was found that too much low-level detail renders the visualizations unclear and hard to read, instead being able to access the low-level information through interactions is a better solution.

In the research of [17], a notation-based visualization technique is proposed (Figure 4.2b), which uses dynamic information on the behaviour of the system. The focus within this research, however, lies more on the design and analysis of said behaviour, rather than the discovery.

In their paper, [19] proposes a Runtime Enterprise Architecture visualization, shown in Figure 4.2c. The width and height of the boxes is scaled by the amount of interaction that that element has. These sizes give a quick overview, whilst the centre of the visualization shows the actual, more detailed, links between the elements.

The fuzzy model proposed by [13], Figure 4.2d, is a Graph-Based technique. In their research, the model is used to explain the violations found using prior in the research. In contrary to most other graph- based methods, the fuzzy model's main information is shown on the edged/links between elements, rather than by changing the visualization of the elements itself.

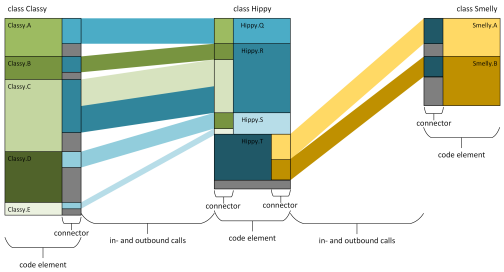
Close to this research is the works of [20] and [24], where a Metaphor-Based Technique using a city to display the various dependencies of the system, shown in Figures 4.2e and 4.2f respectively. The visualization techniques both use a combination of node-link and enclosure for the arrangement of elements, contrary to most other methods. Vizz3D [20] (Figure 4.2e) only displays the code hierarchy and the number of calls made to the respective functions, rather than actual runtime behaviour. In [24] the visualization of behaviour is supported. Additionally, multiple dimensions, such as location, size, and colour, can be changed depending on the information wanted. In their research, it was found that higher level abstractions allow for an easier grouping of the elements, as low-level processes are often only semi-clustered.

4.6 Summary

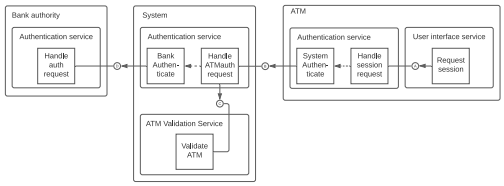
In this chapter, we introduced Software Architecture and its reconstruction from an existing system. It allows us to partially answer our RQS 2 and 3. Software Architecture can be used to convey knowledge about the system to the different stakeholders. This is often done using various visualizations of the views. By reconstructing a Software Architecture, unknown knowledge about the system can be discovered.

We have also seen various visualization techniques, most of which aimed at discovering unknown knowledge. Three categories of techniques could be identified; Graph-, Matrix-, and Metaphor-Based Techniques. Out of these options, Metaphor-Based Techniques are the least constricted in their visual representation, which can include parts of other categories.

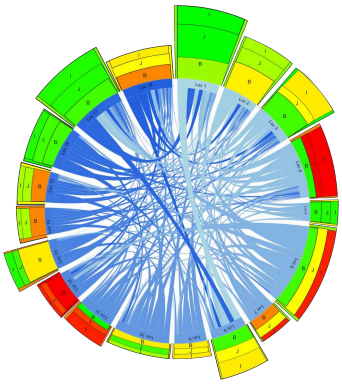
Using Software Architecture does, however, pose various requirements on such techniques and the system itself. Most of the seen techniques require both static input, such as source code of the system, and dynamic input, such as event logs. Techniques that do not require any static input are scarce and do not provide extensive visualizations, such as ArchitectureCity.



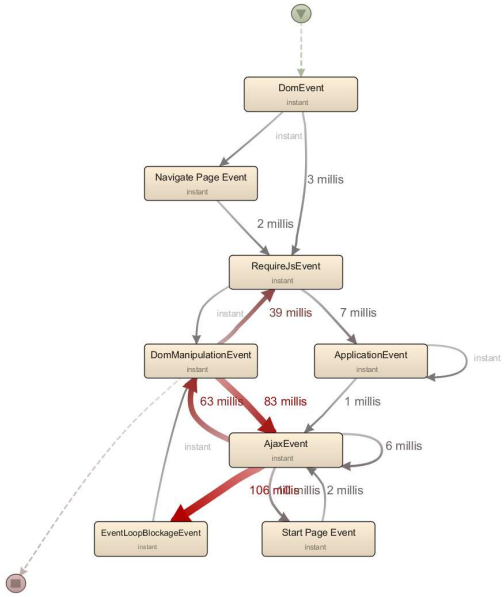
(a) Chord Diagram for a layered architecture, by [14, Figure 8].



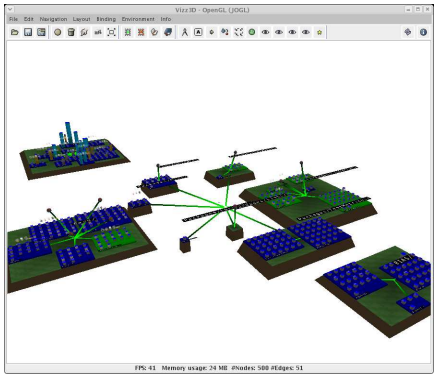
(b) INORA model for ATM running example, by [17, Figure 20].



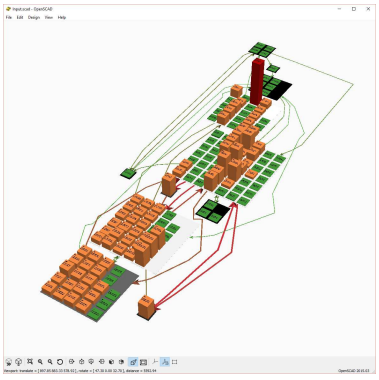
(c) Runtime Enterprise Architecture visualization, by [19, Figure 50].



(d) Fuzzy Model, by [13, Figure 6.14].



(e) Vizz3D Architectural Visualization, by [20, Figure 3b].



(f) ArchitectureCity Visualization using call count based colouring, by [24, Figure 7.2b].

Figure 4.2: Various Software Architecture Visualization techniques.

Chapter 5 Process Mining

Raw data without any processing is hard to obtain knowledge from, a method to transform this data into something useful is needed. Hence, we introduce Process Mining.

5.1 Process Mining

Process Mining sits between the fields of Data Mining and Process Modeling and Analysis, which is also shown in Figure 5.1. Process Mining is defined as [29]:

Definition 5.1, Process Mining [29]:
Process Mining is a method to discover, monitor, and improve real processes by extracting knowledge from event data.

The event data given to the Process Mining method is in the form of event logs, shown on the bottom right in Figure 5.1. An event log is a single dataset containing the event data required to answer a single question regarding the system [29]. Hence, when a large event dataset is available, multiple different event logs can be created to answer multiple questions. The knowledge is extracted from these event logs presented in process models.

The event logs consists of multiple *events*. A set of events belonging to a single common denominator, such as user or object, is called a *trace*. Both events and traces can contain various values, called attributes, of different types associated through a key. In Section 5.3, a deeper look into event logs will be given.

By extracting from the event data, Process Mining establishes a link between the actual process and the data on one hand and Process Models on the other. This is denoted through multiple methods, discovery, conformance, and enhancement, in Figure 5.1. A process model

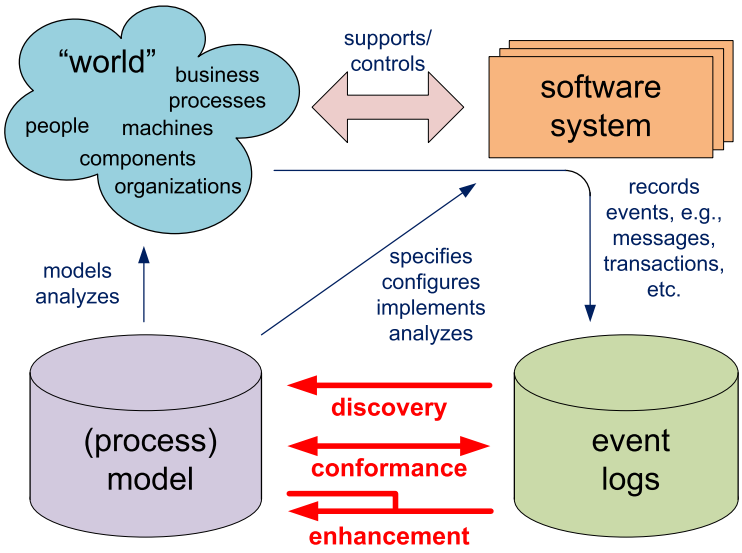


Figure 5.1: Process Mining as proposed by [30, Figure 2.5]. The three main types of process mining: *discovery*, *conformance*, and *enhancement*, are positioned compared to the input log data and the output models.

captures an abstraction of the process, instead of the actual process. The obtained process model can later be used within other processes, such as Business Process Management.

An overview of the full workflow of Process Mining is shown in Figure 5.2. The process starts with the raw event data. The raw event data may come from different sources and locations, can be of many types, and can be assumed not to be well-structured.

The first step with this raw data is to *extract* an event log. This event log should only contain the required information related to the question asked about the system, as not all raw data is relevant information. Selecting only the relevant information, called coarse-grained scoping, is the first action performed in this step. Some questions might only be applicable to a subset of the full system, in which case only information regarding that subset is kept during the scoping. Following the scoping, the data is combined into a single well-structured event log.

This event log is passed to the *filter* step. The filter step is an iterative process, resulting in the filtered event log. In a single iteration an initial analysis of the data is performed, such as finding outliers. With the results of this analysis, the data is filtered such that only the elements of interest to the user remains. Whenever an event log is mentioned without mentioning whether it has been filtered, a filtered event log is implied.

Finally, this filtered event log is given to the mining algorithm. The exact mining process depends on the purpose for which Process Mining is used and the technique chosen. Possible purposes and techniques are discussed in the following sections.

5.2 Purposes

Process Mining finds three main purposes: *discovery*, *conformance*, and *enhancement* of the processes. As shown in Figure 5.1, all three types of Process Mining are positioned between event logs and models. Following [30], the purposes are defined as:

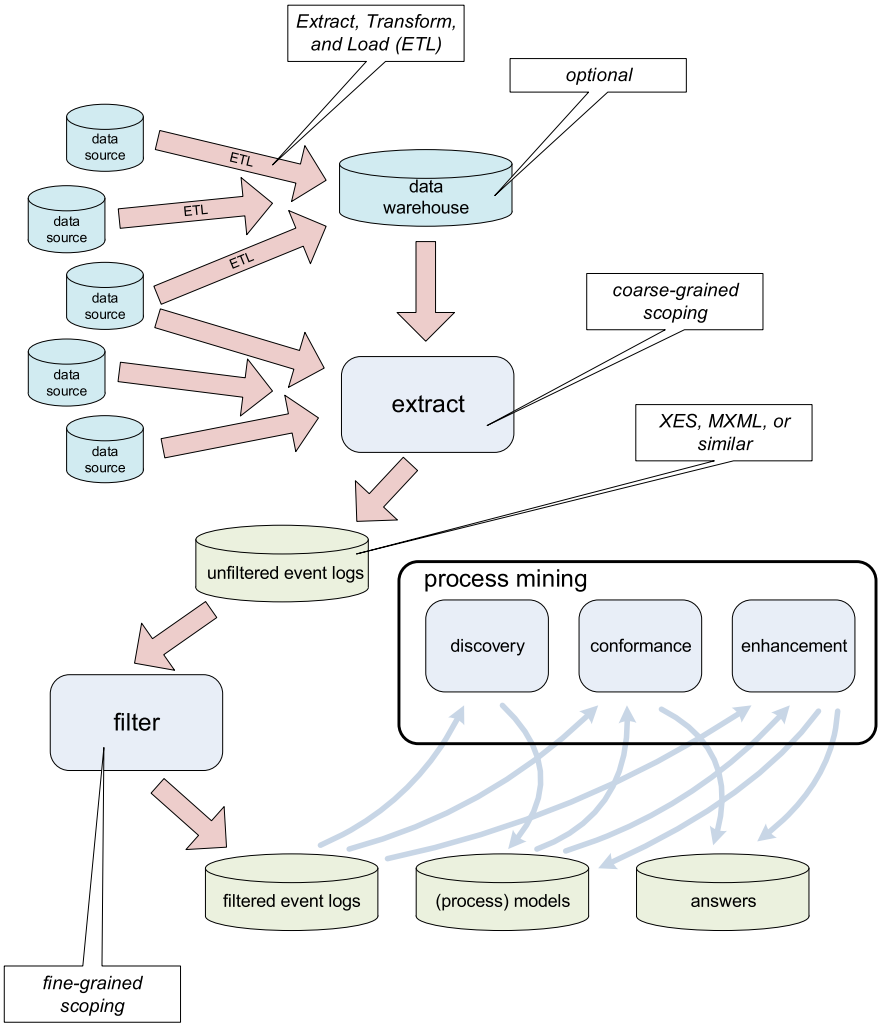


Figure 5.2: Overview describing the workflow of getting from heterogeneous data sources to process mining results, as proposed by [29, Figure 5.1].

Discovery

Process Mining *discovery* techniques create a model from an event log without prior knowledge. This model allows the user to obtain knowledge from a unknown process. When additional information contained within the event log, such as the resource executing the process, this information can be represented within the model, providing additional context. Most of the Process Mining techniques support model discovery [5].

Conformance

Conformance techniques require an existing model of the process as input. The given model is compared to the data in the event log, which allows the user to detect, locate, and explain possible discrepancies between the real process and the given model.

Enhancement

Enhancement techniques also require an existing model of the process as input. Where *conformance* finds discrepancies, *enhancement* tries to extend or improve the existing model with the event data. To types of enhancements can be performed: *repair*, and *extension*. *Repair* changes the model in such a way, that the new version fits the event log better, i.e. fewer discrepancies are present. *Extension* does not change the model, but only adds additional information to it by applying multiple different perspectives when mining the data. Added information can be process performance and bottlenecks.

Techniques for each of these purposes will be discussed in Section 5.6.

The mining process can be performed in an *online* and *offline* setting. In an *offline* setting, all input data is collected and processed prior to the mining process. This means that the size of the data is also known and does not change during the mining process. When mining is performed in an *online* setting, the input data is collected whilst the system is in operation, i.e. it is streamed. The size of the data is thus endless and cannot be processed in the same way as *offline* techniques. By using *online* mining, possible deviations in the process can be detected when they occur, instead of in hindsight. *Online* mining is also referred to as *Operational Support*.

As described previously, current (process) knowledge about the system is assumed to be non-existent, only execution data is at hand. Hence, only the *discovery* purpose of Process Mining can be applied. It allows for transforming the system's execution data to usable process models, which in turn knowledge can be derived from.

Lasagna and Spaghetti Processes

The process model obtained by applying Process Mining can vary on how structured it is. A model can be well-structured, but at the other end of the spectrum unstructured, or anywhere in between, i.e. semi-structured. The names lasagna process for a structured process and spaghetti process for an unstructured process are used by [29], which we will adopt in this research project.

Well-structured, or lasagna, processes their structure is well-defined, i.e. each event refers to an activity in the process, and, in most cases, follows a standard flow. The process has only a small amount of exceptions, i.e. possible paths. This relatively simple process is easier to understand by users and other stakeholders. Activities in lasagna processes can often be automated, due to the well-defined inputs and outputs of each activity. Most Process Mining techniques are able to process lasagna processes.

Between well-structured and unstructured are the semi-structured processes. Here, the process itself is well-defined, but activities can deviate from this standard process or need judgement of a human.

Unstructured, or spaghetti, processes are the opposite of well-structured processes, where it mostly does not have a clear structure and/or follow a standard flow. An example of such

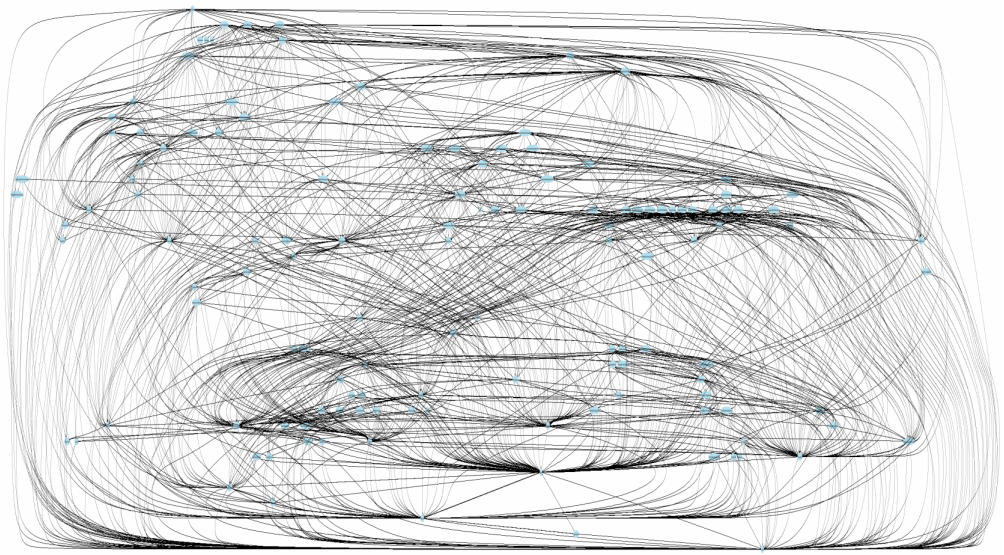


Figure 5.3: An example of a spaghetti-like dependency graph, as shown by [24, Figure 3.6].

unstructured process is shown in Figure 5.3. Not all Process Mining techniques are able to deal with such unstructured event logs. Hence, special techniques, tuned towards such processes, need to be selected, which might limit the user in the discovery process of the system.

A formal distinction between well-structured and unstructured processes is impossible. As a rule of thumb, [29] uses the following: a process is a lasagna process if with limited efforts it is possible to create an agreed-upon process model that has a fitness of at least 0.8, i.e., more than 80% of the events happen as planned and stakeholders confirm the validity of the model.

5.3 Event Data

As stated in Section 5.1, an event log is a single dataset containing the event data required to answer a single question regarding the system [29]. An event log consists of *Events*, which denote the execution of one, or possibly more, activity within the process. Each *Event* contains several *Attributes* which contain the actual information of this event. An example of such attribute is the name or id of the activity performed in this event.

Some common attribute types are the process identifier, timestamp, and activity identifier. The process identifier is some type of identifier denoting to which process instance the *Event* belongs. Each event occurs at some point in time, which is often denoted through either a timestamp or identifier, such as the unix timestamp, which denotes the (partial) ordering of the events. The activity identifier denotes which activity happened at the given timestamp within the given process instance.

The event log can be stored in multiple ways, such as in a database or serialized to a file. The two most predominant formats in which the data is serialized, for file-based storing, are XML and JSON. Additionally, two standards are defined to structure this data: XES [31] and OCEL [21].

The XES standard is *Trace*, or *case*, centred, where a trace embodies a path taken by a single execution of the process. This is often denoted through an identifier attribute referring to the unique process instance. A XES log serialized using XML is shown in Example 5.1. Another

example of a trace in the context of a car-dealership could be a customer purchasing a new car, where the identifier is the invoice number.

In the OCEL standard the focus on traces is removed, but rather focuses on different objects. This allows the log to contain multiple case notions, which often occurs in real life data such as the SAP ERP [21]. An OCEL file, also serialized with XML, is shown in Example 5.2.

```
<?xml version="1.0" encoding="UTF-8" ?>
<log xes.version="2.0" xes.features="arbitrary-depth"
  xmlns="http://www.xes-standard.org/">
  <trace>
    <string key="concept:name" value="Trace number one" />
    <event>
      <string key="concept:name" value="Register client" />
      <string key="system" value="alpha" />
      <date key="time:timestamp"
        value="2009-11-25T14:12:45:000+02:00" />
      <int key="attempt" value="23" />
    </event>
    <event>
      <string key="concept:name" value="Mail rejection" />
      <string key="system" value="beta" />
      <date key="time:timestamp"
        value="2009-11-28T11:18:45:000+02:00" />
    </event>
  </trace>
</log>
```

Example 5.1: An example XES file containing only trace, events, and non-nested attributes [31] serialized as XML.

5.4 Multi-Process Mining

Most processes in practice involve multiple inter-related entities [9]. This means that entities within a system's process do not only belong to a single process at a time, instead multiple processes work with the entity at a single time or the entity is passed between processes and back. The event logs used in a typical process mining technique are so-called sequential event logs, where the events belonging to a single case from a sequence. The most used sequential event log standard is XES. These sequential event logs are unable to contain the inter-related information.

Multi-Process Mining tries to tackle the problems that arise in order to deal with these inter-related entities and processes. Following [10], the problems can be split in a quadrant, as shown in Figure 5.4. These quadrants are:

Single Executions, Single Object Each process execution relies on its own single entity and is not influenced by other processes, this is classical Process Mining.

Single Executions, Multiple Object Each process is not influenced by other processes, but relies on multiple different entities. This results in a dependency between the different

```
<?xml version="1.0" encoding="UTF-8" ?>
<events>
  <event>
    <string key="id" value="e1" />
    <string key="activity" value="place_order" />
    <date key="timestamp"
        value="2020-07-09T08:20:01.527+01:00" />
    <list key="omap">
      <string key="object-id" value="i1" />
      <string key="object-id" value="o1" />
      <string key="object-id" value="i2" />
    </list>
  </event>
</events>
```

Example 5.2: An example OCEL file [21] serialized as XML.

entities, as they might require each other in order to continue in the process. An example of such cases are ERP and Supply Chain systems.

Multiple Executions, Single Object When multiple processes execute, but with each their own entity, dependencies between the different processes are present. A call centre is an example of a Multiple Executions, Single Object system. In a call centre many employees are working with a single client at a time, here each employee is seen as a single process instance. When the employee cannot solve a particular problem, the client is transferred to a different employee who can. Here, the process depends on the execution state of another process and both contain the same entity.

Multiple Executions, Multiple Object In the most complex systems, dependencies are formed between both the processes and the entities. In these cases the techniques required of the previous two cases are both used to solve the combined problem.

A proposed method which can store the non-sequential event data is the use of database systems to store this data. However, with the usage of database systems, obtaining the actual information related to a single case is a much harder task. Research regarding this topic and to solve the storing and querying problems are present [9], but still in its infancy. One standard, however, has been proposed: OCEL, which was described in Section 5.3. Hence, this research project will be scoped to event logs containing only sequential data, belonging to the Single Executions-Single Object quadrant.

5.5 Requirements for Process Mining Visualization Techniques

Process Mining is impossible without proper event logs [29]. Hence, Process Mining poses requirements on the input data before mining. Additional requirements may be present, but these depend on the mining technique chosen, such as needing a timestamp.

Minimal Subset (Non-Functional)
The event log contains only the needed information to answer the user's question regarding

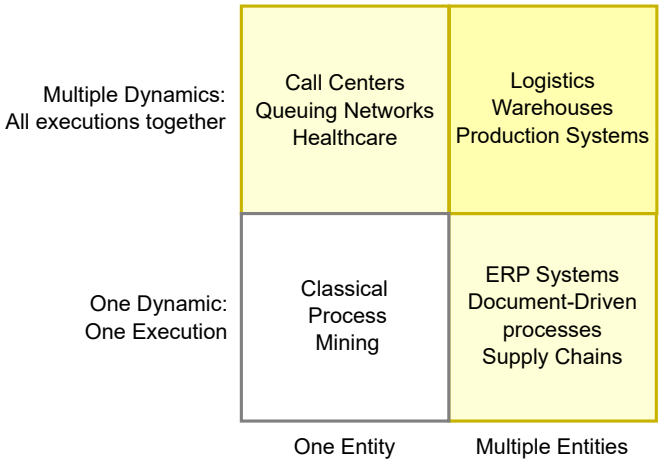


Figure 5.4: Multi... Process Mining quadrants, as shown by [10].

the system.

One-to-One relation (Functional)
An event log corresponds to exactly one process. Each event contained in a log should correspond with exactly one process instance, referred to as *case*.

Ordering (Functional)
Events within a single case should be orderable.

Required Data (Functional)
Each event should contain at least a case and activity. The additional information is referred to as attributes.

5.6 Process Mining Visualization Techniques

After an event log is mined, the results can be visualized using varying techniques and algorithms. In this chapter we will look at some of these visualization techniques. In order to classify the different visualization techniques, we will use the same classifications used in Software Architecture, described in Section 4.5. These three classifications are: Graph-, Matrix-, and Metaphor-Based Techniques.

Additional to the visualization technique classification, a distinction between two types of representations can be made: aggregated instances as a whole process or the individual process instances. Representing the aggregated process allows the user to obtain an overview of the process as a whole and the interactions between activities. Representing process instances, on the other hand, provides better insights into anomalies and violations. A combination of the two kinds is also possible, where the entire process is annotated with process instances.

The research that defines Process Mining, [29], also lists several visualizations techniques. Their research starts with Notation- Based (i.e. Graph-Based) Techniques, representing the process itself; BPMN (Figure 5.5a) and Extended Petri-Net (Figure 5.5d). The Extended Petri-Net uses additional annotations, such as employee roles for an activity, to provide more information.

A similar concept can be seen in Figure 5.5b, in which the runtime duration of each activity is shown as a bar-chart in 3D. In their data, four groups are identified and each group is given

its own bar. The size of the group is reflected in the width of the respective bar, and since the grouping does not alter during the process, this width does not change per activity. The height of the bar determines the actual runtime of that activity for that group. By using a colour to denote the group type and using the dimensions for both the size and runtime, the user can make quick and intuitive comparisons between the different activities and groups.

Following the Notation-Based visualizations, [29] also proposes Graph- and Matrix-Based visualizations. Figure 5.5e shows the interactions between activities as a social network, i.e. a Graph- Based technique displaying the aggregated process as a whole. Each link between two nodes denote an interaction, where the width of the link denotes the frequency of this interaction. Alternatively, a Matrix-Based instance visualization can be made, shown in Figure 5.5f, which allows for an overview of the distribution of process runtime, which was is lacking in the Graph-Based visualization (Figure 5.5e).

Similar to Figure 5.5f, [12] proposes dotplot visualizations, a Matrix-Based interaction visualization. In the visualization, all activities are represented on both the x- and y-axis, the cells show the association between the two activities, producing a pattern of colours. These patterns imply different types of behaviour within the system, from which some common patterns are identified. Visually matching these common patterns with the found matrix, allows the user to identify certain sub-parts of the system or assess the system as a whole.

The research of [15], focuses on the extraction of Software Architecture Reconstruction based on the dynamic information found in event logs. The technique proposed is a Notation-Based technique, shown in Figure 5.5g, representing the aggregated process as a whole. Most notably, the visualization groups elements in a hierarchical manner, similar to the enclosure method explain in Section 4.5. The activities themselves and links between activities are not grouped, providing detailed information on the interactions.

Alternatively, [12] applies a dimensionality reduction method on the event data, such as the Self-Organizing Map (SOM). The Self- Organizing Map technique uses artifical neural networks to map high-dimensional data to low dimensional spaces [12] With a greatly reduced number of dimensions, more alternative visualization techniques can be used, one of which is shown in Figure 5.5i. Reducing the number of dimensions requires data of the process as a whole and thus is used for aggregated process visualizations. Although allowing for possibly easier and alternative visualization techniques, it does not allow for deep insights into the system, as this information is lost. Instead, it provides an overview and can highlight interesting areas of the process.

The fuzzy model (Figure 4.2d) previously described in Section 4.5 was initially proposed by [12] with the purpose of Process Mining. This model is a Graph-Based technique showing the frequency of interactions with the size and colour of the links between elements.

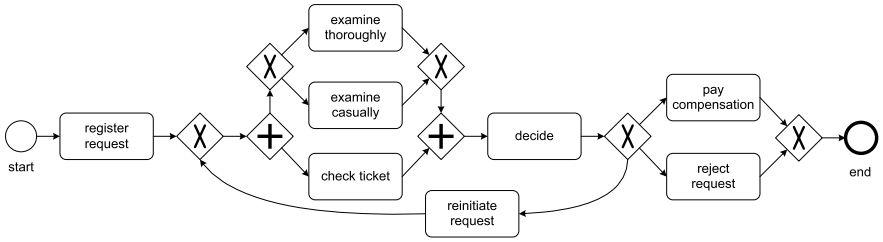
Lastly, [3], shown in Figure 5.5c, propose a method to visualize traces with the possibility to zoom in on a selected part of the traces. The visualization is closest to a Matrix- Based technique.

5.7 Summary

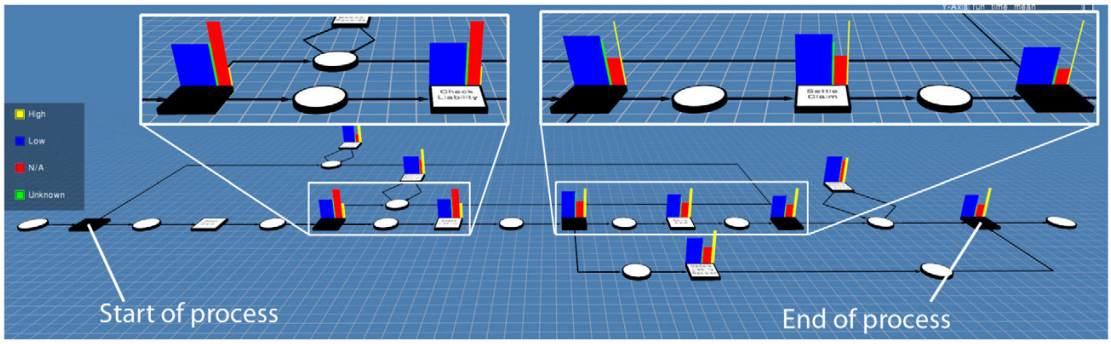
In this chapter Process Mining is introduced. Process Mining is used to transform raw dynamic event data into insights on the actual process. It can fulfil multiple purposes, out of which discovering new insights in the actual process will be the main role within this research project. Additionally, Process Mining assists in answering RQS 2 and 3.

To allow for effective Process Mining, requirements are set on the input event log, such as an ordering of events within a single case. A Top-Down approach needs to be taken, as no additional input is given other than the event log, as stated in Chapter 1.

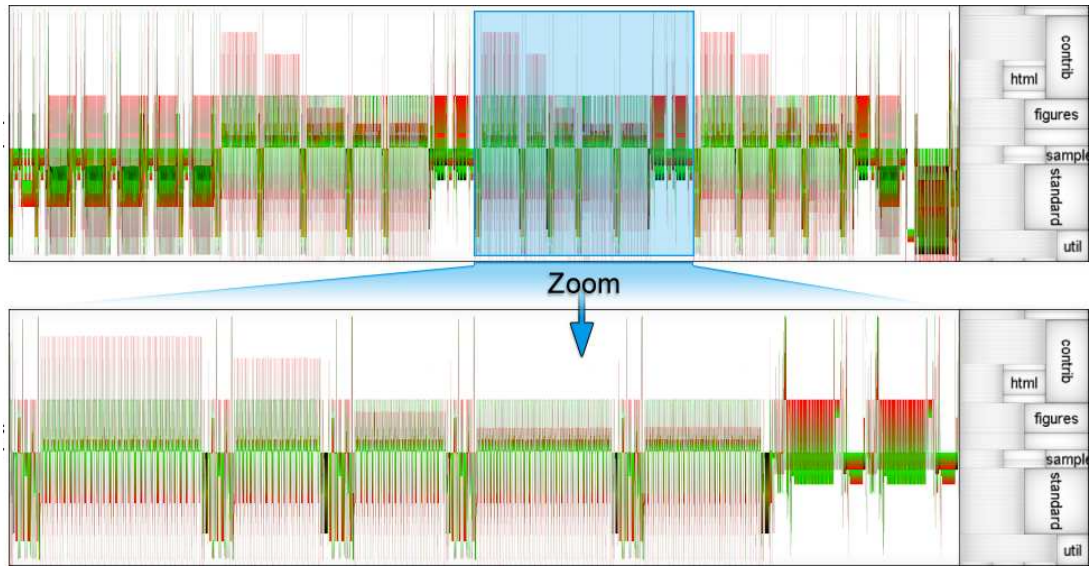
Two different types of visualizations could be identified; aggregated process visualizations and



(a) A BPMN modelling the handling of compensation requests, by [29, Figure 2.3].

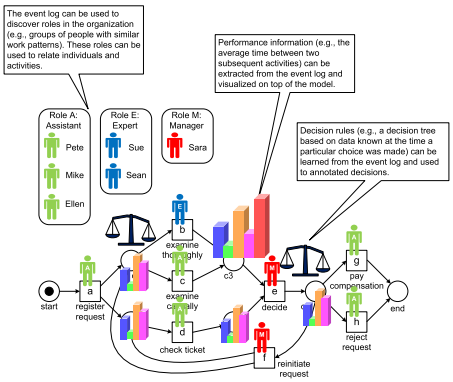


(b) A runtime visualization using ProcessProfiler3D, by [35, Figure 11].

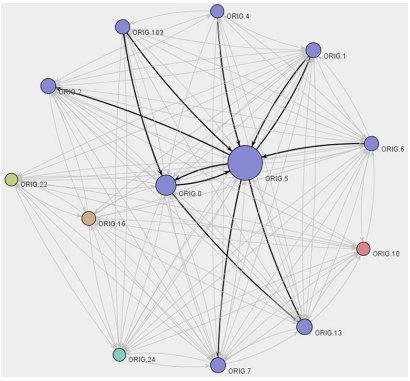


(c) Trace visualization using JHotDraw zooming in on a selected feature, by [3, Figure 4]

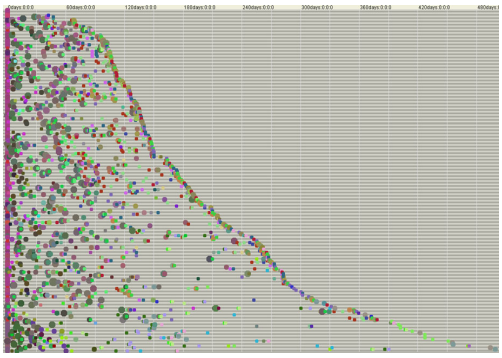
Figure 5.5: Various Process Mining Visualization techniques.



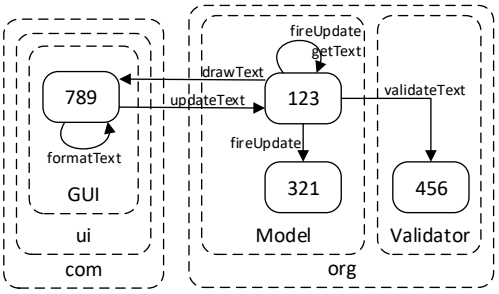
(d) An extended petri-net process model of the Figure 5.5a process, by [29, Figure 2.8].



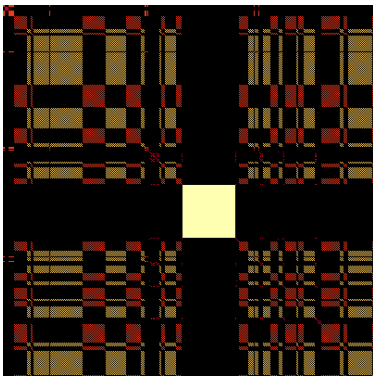
(e) A Social Network graph based on the handover of work, by [29, Figure 13.9].



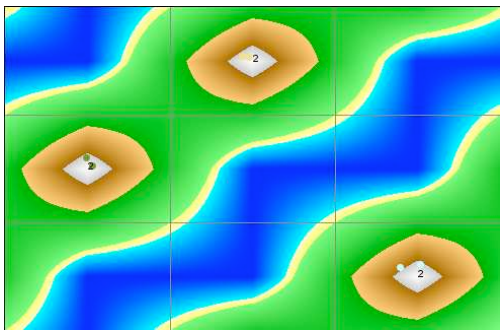
(f) A dotted chart with each event in relative time, by [29, Figure 9.4].



(g) Hierarchical Interaction Model, by [15, Figure 2].



(h) Dotplot visualization of a highly repetitive log, by [12, Figure 8.7].

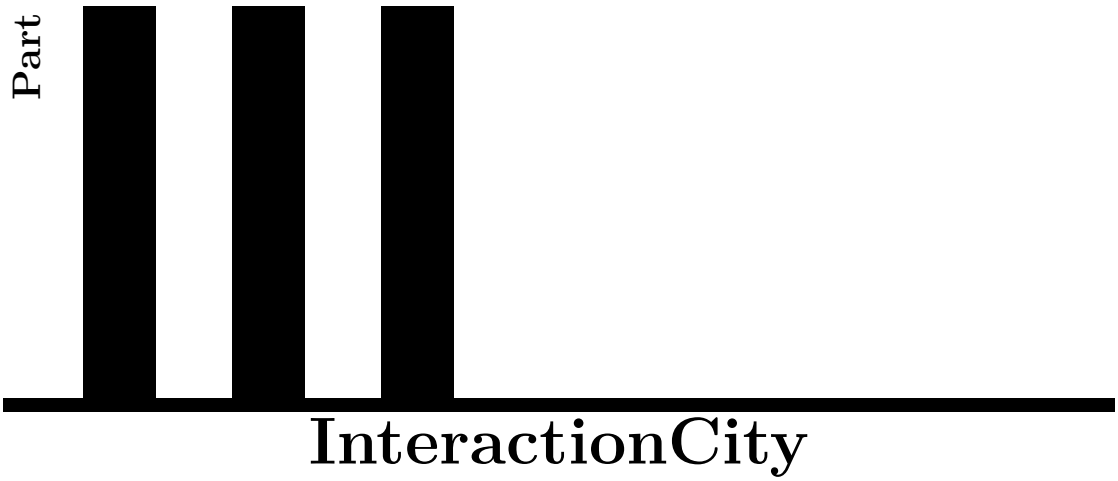


(i) A Self-Organizing Map (SOM) based event log reduced to two dimensions, by [12, Figure 6.17].

Figure 5.5: Various Process Mining Visualization techniques. (continued)

process instance visualizations. Aggregated visualizations allow for assessment of the process as a whole, whereas instance visualizations allow for better anomaly and violation. Both types can also be combined, where the process instance information is added to the process as a whole.

Various visualization technique implementations are shown, most of which are Notation-Based for aggregated process visualization. For process instance visualization Matrix-Based techniques are often used, required due to the number of elements in the data. Techniques using a, more intuitive, Metaphor- Based approach are lacking or do not allow the discovery of the system, but instead are focussed on the presentation of additional data for analysis of the system.


Part III
InteractionCity

Chapter 6 Solution Design

In this chapter we specify and elaborate on the design of our new visualization technique: InteractionCity. The complete overview of the framework is depicted in Figure 6.1. The proposed framework builds upon Visual Data Exploration, as shown in Figure 3.1. In the overview figure, the conceptual elements are replaced with actual elements used in that step. Our proposal uses interaction data (see Section 6.2) from which an interaction model is derived ("mined", see Section 6.3). This model is then visualized so that the user can explore the interactions in the log (see Section 6.4). At the end of the process, the user has three main artefacts: a Network model, a Map visualization, and a World visualization. Each of these artefacts provide the input for the following step. Contrary to Figure 3.1, some lines are dotted instead of solid. This implies that the lines are present in Visual Analytics itself, but will not be (fully) used in the technique presented in this work.

The last step in the process, obtaining architectural knowledge from the visualization, is not automated and should be performed by the user solely instead. With the visualization the user can apply various techniques, including those from Software Architecture, to obtain the context knowledge needed.

To explain the approach in the remainder of this chapter, we use the running example presented in Section 6.1.

6.1 Running Example

For the following sections we use an online TODO list as running example. In this program, each user has only a single TODO list. This list is shown through both a mobile app and a browser webapp, both allow the user to modify the list in various ways. Since this list can be managed through multiple frontends, it is stored on a central backend. Additionally, the app uses a local Model-View- Controller model, in which the view and underlaying local data are decoupled, to

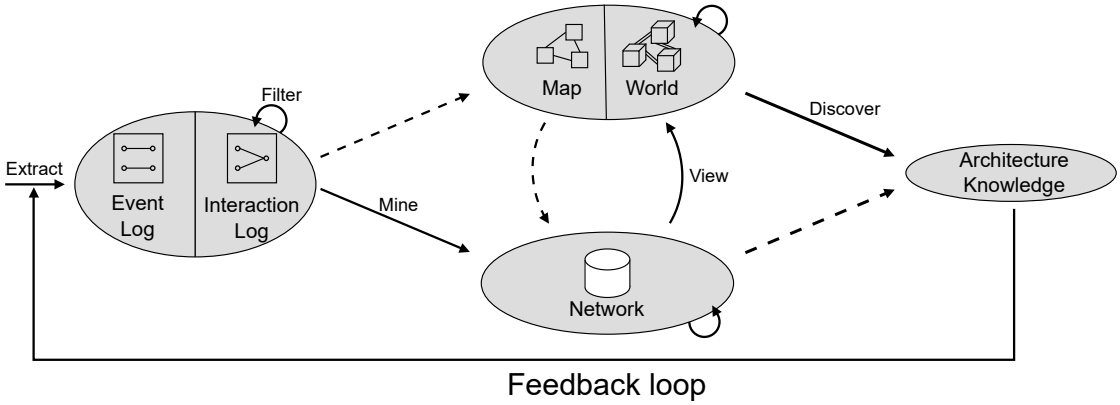


Figure 6.1: Visual Data-Exploration for the InteractionCity Technique.

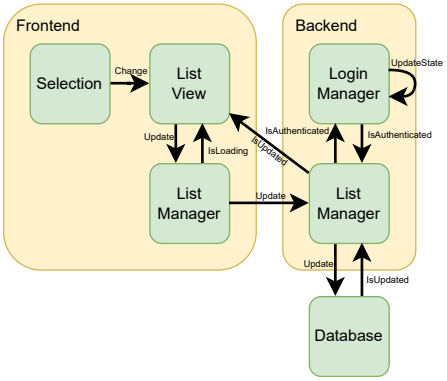


Figure 6.2: A Component Diagram of the running example.

allow for a better user experience. For any of the interactions with the backend, the backend replies with a success state, such that the user is always aware if their modification or request has been handled properly. This backend performs an authentication of the user, such that other users cannot view or modify other TODO lists.

The Component Diagram of the various classes present in the system is shown in Figure 6.2. In the system various elements can be identified: Selection, ListView, Frontend ListManager, Backend Listmanager, LoginManager, and Database. Additionally, arrows between these elements show a subset of the possible functions that can be used to interact between to elements. In this diagram, the frontend is connected to only a single backend, however if the service ever needs to scale the backend server can be duplicated and users distributed over the different servers, i.e. load-balanced.

For the running example, we will look at a single, successful, usage case “U1”: an update of a set of TODO items. The Message Sequence Chart of the U1 case is shown in Figure 6.3. A user has a set of TODO items selected, stored in the Selection element, which are modified. This modification is chained through to the ListView and ListManager, which in turn pass the change to the backend. In the backend the ListManager receives the change. It first checks whether the user is properly authenticated, which updates the last-login state of the user, and when done so passes the chance through to the database for persistent storage. Once this change is stored, a

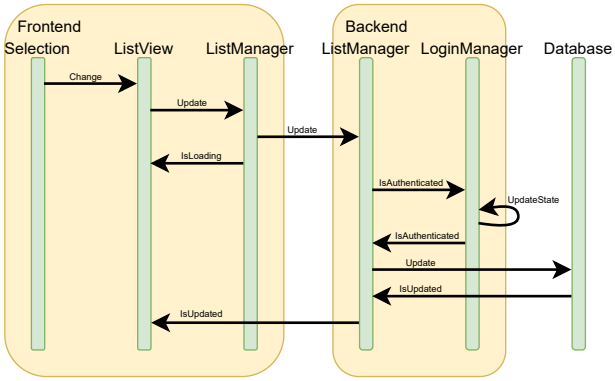


Figure 6.3: An example Message Sequence Chart for a single case `U1`.

message is sent back to the frontend's ListView, to notify the user of the successful change.

Lastly, the Fully Qualified Names (FQNs) of all the elements, in order of the Message Sequence Chart, are shown in Example 6.1. The company which develops this TODO app is an international company named "Example" and the project itself "TodoApp", which is reflected within the FQNs. Additionally, the first two events within the raw event log are shown in Example 6.2.

```
com.example.TodoApp.Frontend.Selection
com.example.TodoApp.Frontend.ListView
com.example.TodoApp.Frontend.ListManager
com.example.TodoApp.Backend.Listmanager
com.example.TodoApp.Backend.LoginManager
com.example.TodoApp.Outsource.Database
```

Example 6.1: The Fully Qualified Names of the system shown in Figure 6.2.

6.2 Interaction Data

As described in our design objective, defined in Chapter 1, the input data is the system's execution data, which was further defined in Section 5.3. This section lists the requirements and concepts of the Interaction Data, which relates to the first step of the process shown in Figure 6.1. In our running example the raw input data is not specifically given, but used as a illustrative example instead.

On the data format used for the input data, no requirements are set, such that these can be changed depending on implementation and system context. In classical Process Mining the number of hard requirements set (Section 5.5) are limited, for InteractionCity some additional requirements are needed. Before these requirements can be set, two concepts are to be introduced: Interaction Log, and Fully Qualified Name Hierarchies.

```
...
<trace>
  <string key="case" value="U1" />
  <event>
    <string key="timestamp" value="2022-10-01T10:00:01" />
    <string key="location" value="com.example.TodoApp.Frontend.Selection" />
    <string key="action" value="send" />
    <string key="message" value="Change" />
  </event>
  <event>
    <string key="timestamp" value="2022-10-01T10:00:02" />
    <string key="location" value="com.example.TodoApp.Frontend.ListView" />
    <string key="action" value="receive" />
    <string key="message" value="Change" />
  </event>
  ...
</trace>
...
```

Example 6.2: First two events of the Running Example Event Log

6.2.1 Interaction Log

In Chapter 5, the notion of cases and activities are explained: the events each belong to a case and belong to a single activity. This works well when each case, event, and activity act on a single object only. This, however, is not always true.

To omit this issue, rather than looking at one activity that occurs at one point in time, the focus is shifted to the interactions. An interaction, compared to an event, does not occur on a single object or activity, but rather are between objects or activities. Hence, an interaction does not contain a single activity, but a source and a target object. Additionally, these interactions can contain other data, such as a message being sent or an object being passed to the other party. An example of an interaction is notifying to another service of an event. An interaction log is similar to an event log for events; interactions belong to a single case and are stored in an interaction log. Thus, the notion of case within a log is carried over from classical process logs.

As the base of the Interaction Log a directed hierarchical multi-graph. A multi-graph, contrary to a classical graph, allows multiple edges to have equal connected nodes, i.e. a node can be connected to a single other node multiple times. A hierarchical graph, compared to a classical graph, allows nodes to be nested within another. The edges can also be connected to the nested nodes, however these edges cannot extend beyond a nested node's parent, i.e. there exist only intergroup edges. A metamodel of an Interaction Log is shown in Figure 6.4.

A similar method for interactions is a Message Sequence Chart, of which an example is shown in Figure 6.3. The objects interact with each other, visualized with the arrows. In the running example an interaction would be the check "IsAuthenticated" performed by the backend's ListManager with the LoginManager. Message Sequence charts are only for a single case, resulting in limited use for the, possibly large number of, events provided in the input data.

Contrary to activities, interactions do not have a single timestamp, as they do not occur at a single point in time. Instead, an interaction can have a duration or a start and end timestamp

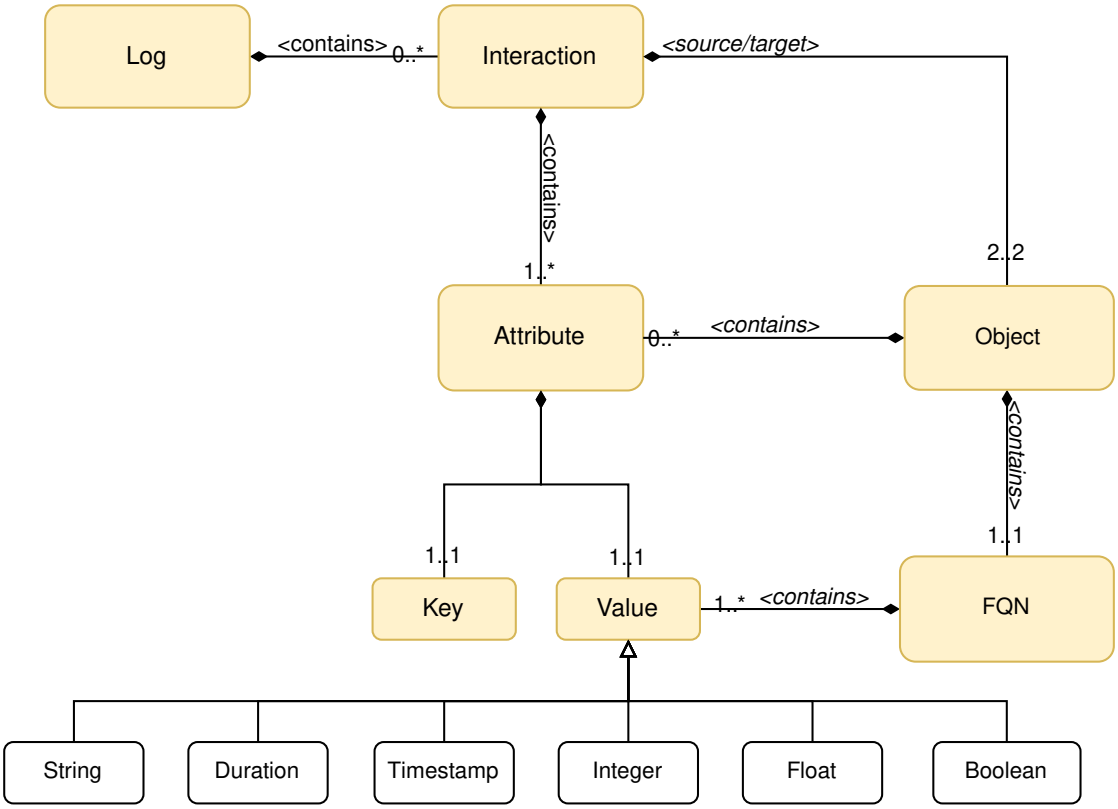


Figure 6.4: A metamodel of an Interaction Log.

from which the duration can be derived.

As with Process Mining, we can also set requirements for Interaction Mining:

Minimal Subset (Non-Functional)
The interaction log contains only the needed information to answer the user's question regarding the system.

One-to-One relation (Functional)
Each interaction contained in a log should correspond with exactly one process instance, referred to as *case*.

Ordering (Functional)
Interactions within a single case should be partially ordered.

Required Data (Functional)
Each interaction should contain at least a case, source object, and target object. The additional information is referred to as attributes.

Optional Data (Functional)
Each interaction can contain a duration or a starting and ending timestamp, from which the duration can be derived. When a starting and ending timestamp is given, the ordering can be resolved using these timestamps.

The raw input data of the first two events in the event log are shown in Example 6.2. Using this event log, a simple Interaction log in the XES format is made. This full Interaction Log is shown in Appendix A, of which the first interaction is shown in Example 6.3. In this interaction log some additional keys are present; `location_org`, `location_project`, `action`, and `message`. These keys will not be used for the InteractionCity.

Obtaining such a dataset, depends on the application. For a system which logs interactions rather than events, creating an interaction log will only be some, relatively simple, transformations to obtain an XML formatted dataset with the needed attributes. When only an event log, as is the case in this running example, is available, an interaction log can be created by connecting each event to the following event. The first event provides the information of the interaction's source, whilst the following event provides the information of the interaction's target. The first and last events of a trace do not have a previous or following event, and hence will not be used twice; both as source and target.

6.2.2 Fully Qualified Name Hierarchies

The elements within the input execution data, e.g. activities in events logs, each have an identifier. The event log activities, especially those from software systems, can be in the form of a fully qualified name (FQN). A fully qualified name is an identifier for an element within the log which uniquely specifies that element even when the log itself does not provide any context. An example of an FQN in a software system is: `Organization.Project.Namespace.Class.Function.instance`.

A FQN consists of the instance's identifier itself and any parent elements their identifiers. So although any context knowledge of the data is missing, the FQNs contain parts of the hierarchy of the system. In classical Process Mining this hierarchical information is left untouched. By using the identifiers as individual parts, the hierarchy can be recreated.

In Figure 6.3, the example messages are given and the frontend and backend groups have been created around the elements. The image highlights the Message Sequence Chart of only a single case, `U1`. The accompanying FQNs are also listed in Example 6.1. By using these names to

```
...
<string key="source:timestamp" value="2022-10-01T10:00:01" />
<string key="source:location_org" value="com.example" />
<string key="source:location_project" value="TodoApp" />
<string key="source:location_namespace" value="Frontend" />
<string key="source:location_class" value="Selection" />
<string key="source:action" value="send" />
<string key="source:message" value="Change" />
<string key="target:timestamp" value="2022-10-01T10:00:02" />
<string key="target:location_org" value="com.example" />
<string key="target:location_project" value="TodoApp" />
<string key="target:location_namespace" value="Frontend" />
<string key="target:location_class" value="ListView" />
<string key="target:action" value="receive" />
<string key="target:message" value="Change" />
...
```

Example 6.3: First interaction of the Running Example Interaction Log

recreate the hierarchy, we obtain the Component Diagram shown in Figure 6.2. This diagram shows both the hierarchy of the system and the original messages being sent to each other.

6.2.3 Data Requirements

On the input data for a InteractionCity, several requirements are set. The first of these requirements it that the input data is an interaction log. Interaction logs come with their own set of requirements, which are inherited. In an Interaction Log an additional duration, or start and end timestamp from which the duration is derived, can be given for each interaction. For InteractionCity this duration value is a required rather than optional field.

Each of the interactions contains a source and a target identifier, for InteractionCity these both need to be Fully Qualified Names (FQN). A FQN has a certain number of parts which can vary for each interaction and can even vary between the source and the target of the interaction. For the input data of InteractionCity, each interaction can have a different number of parts, however each source and target belonging to the same interaction need to have an equal number of parts.

Additional data, attributes, may be present in the execution data, but will not be used in an InteractionCity.

To summarize, for InteractionCity, requirements on the input data are set, additionally to the requirements set in Section 5.5:

Log (Functional) The log should be an Interaction Log

Interaction (Functional) Each interaction should contain a duration.

Identifier (Functional) Each identifier should be a Fully Qualified Name.

Interaction Identifiers (Functional) An interaction source and target identifier should have an equal number of parts.

Ordering (Functional) Interactions within a single case should be partially ordered.

6.2.4 Event Log Transformation

Whilst obtaining an interaction log from the system directly, such log is not always available. Instead, a classical Event Log can be is more likely to be available, or derivable from the existing data. Hence, a method to transform the classical Event Log into an Interaction Log is needed. To perform this transformation, we propose a quick and simple technique, which does lack validation or additional features to ease the transformation.

To obtain the interactions from the events, each event should be connected to the next event within its trace. This assumes that the events within each trace are sorted, based on occurrence or timestamp of the event. The process starts by creating pairs of events, the source and target event, where the nth pair contains the nth event as source and the nth plus one event as the target. Since the last event in the trace has no following event, the number of interactions is one less than the number of events in the log. This results in all but the first and last event being used once as source and once as target.

The values of the both the source and target event are added to the interaction. In order to distinguish between the values, the keys of each event are prefixed with "source:" and "target". In Example 6.4, both input and output example values are given, when using the transition plugin.

6.3 Interaction Network

A model contains information derived from the original dataset, which can include both the raw data and data aggregated from the raw data, depicted in the Figure 6.1 as the second step. To derive the model, the input data is mined. In an InteractionCity, the raw data is not needed once the model is created. In doing so the raw data only needs to be processed once, which allows for larger datasets. Additionally, the model can be shared without requiring the raw input data. Hence, the connection between Interaction Data and InteractionCity Visualizations in Figure 6.1 is shown with a dashed rather than solid line.

A Component Diagram, for example the one shown in Figure 6.2, only displays the hierarchy for a single case. In order to obtain knowledge of the system as a whole, the model in our technique needs to allow the user to obtain knowledge on both the hierarchy and all cases. Additionally, information on the interactions, connecting two objects, need to be captured in the model.

Hence, the Interaction Network uses an Interaction Log as its base (Section 6.2.1). However, rather than using a directed hierarchical multi-graph, undirected edges are used. These edges are, thus, also aggregated. Aggregating the edges, rather than representing each individual edge by itself, greatly simplifies the total model, helping the user. However, the undirected edges will still need to hold aggregated information on both directions, as this information is still relevant for the user.

Some edges are looping edges, i.e. have the same node as both the source and target. Rather than storing the aggregated information as such edge, this information is stored within the node itself and no looping edge is stored. Additional to this looping information, each node also stored aggregated information on all the incoming and outgoing edges, i.e. edges where this node is either target or source, respectively.

To obtain the Interaction Network, the raw input data is mined, which is performed in multiple steps. The mining process starts by obtaining the hierarchy for the model, rather than the edges. Each of the interaction in the input data contains a source and a target identifier. This identifier is, as set in the requirements, a Fully Qualified Name. Each of the individual parts of the identifier

```
Event Log (input):
-----------------------------------------------------------------
log [
  case [
    event {
      activity = frontend.listview
      action = send
      message = update
    },
    event {
      activity = frontend.listmanager
      action = receive
      message = update
    }
  ]
]


Interaction Log (output):
-----------------------------------------------------------------
log [
  transition {
    source:activity = frontend.listview
    source:action = send
    source:message = update
    target:activity = frontend.listmanager
    target:action = receive
    target:message = update
  }
]
```

Example 6.4: Example input and output of a simple transformation from Event Log to Interaction Log

forms a node within the model, in which duplicate identifier parts are ignored. For example the FQN `Project.Class.Method` would result in three nested nodes; `Project`, `Class`, and `Method`.

With the full hierarchy made, the edges can be added to the model. Here, each interaction forms an edge between the nodes denoted through the source and target identifiers. The interactions themselves are both not aggregated and directional, hence these are further processed. First, the set of interactions between from a node to another are aggregated into a single directed edge. Nodes between two directed edges are present can then be combined into a single undirected edge.

Within the input data, the interactions can be between two nodes that do not belong to the same hierarchy group. To solve this issue, the intra-group edges are reconnected to the parents of its source and target nodes, such that the both nodes belong to the same group.

The Interaction Network provides us with the aggregated information of interactions, however what information to aggregate is yet undefined. Since the model should function for unknown contexts and interaction logs with unknown attributes, aggregation can only be done on aspects which are independent of the unknown log attributes. As set in the requirements, only the source, target, case, and duration of each interaction is known to be present. The duration can be aggregated using various methods. An example of an aggregation on duration is the minimum and maximum duration for a set of interactions. Other than the known attributes, generic information can also be aggregated, such as the number of interactions between source and target.

6.4 Visualizations

In Figure 6.1, the third step is the visualization step. The input for this step is the model, Interaction Network, previously obtained. This visualization step does not modify the model by itself and thus the arrow providing input back to the model step is shown as a dashed line. This step, however, can only partially show the nodes and edges in the model.

Let us start with looking back at our design objective, as described in Chapter 1:

Improve the architectural and context knowledge *by* creating a new visualization technique *that* uses the system's execution data *in order to* improve the understanding of said system.

As the (context) knowledge about the system is lacking, the identification of visual elements and mapping between visual elements and system elements should be intuitive. Hence, a Metaphor-Based visualization technique, as described in Section 4.5, was chosen. The selected metaphor is a city, similar to ArchitectureCity [24] and Vizz3D [20]. In the further specification of the visualization technique most visualization dimensions take analogies similar to the real world. An example of the proposed visualization technique is shown in Figure 6.5.

Using the city metaphor, two different visualizations are made; the Interaction Map and Interaction World visualization. Both visualizations show the same city, but differ in the amount of detail shown to the user.

The Interaction Map shows the city similar to a real-world map. This is a top-down view of the city, focusing on the positioning of various visual elements, rather than actual elements themselves. In most visualizations, positioning takes a key role for the user to better obtain knowledge from the visualization. To further allow the user to tune the visualization to their needs, the positions of the elements on the map should be editable. The Interaction World, on the other hand, shows the city in a 3D manner. In this visualization the visual elements their details are the focus, whilst the positioning only provides visual aid.

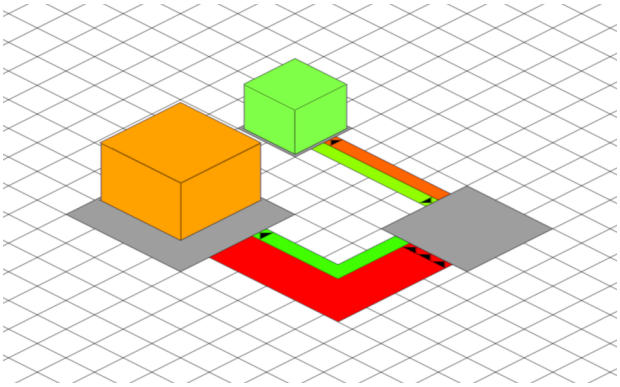


Figure 6.5: An example of the proposed InteractionCity visualization.

Visual Dimension		Input Dimension
Street	Width	Maximum number of interactions
	Colour	Maximum duration of interaction
Neighbourhood	Colour	Abstraction level depth
Building	Width	Number incoming and outgoing transitions
	Height	Number of unique cases
	Colour	Maximum duration of looping-transitions

Table 6.1: Example mapping between aggregated process data, visualization dimension, and metaphor element of InteractionCity.

In the city metaphor, three (visual) elements can be found: streets, buildings, and neighbourhoods. The interactions themselves can be thought of as the cars driving on the various street from building to building within various neighbourhoods. These cars (i.e. interactions) themselves do not form a visual element. The Interaction Network edges are represented though streets, which link to neighbourhoods and buildings. Naturally, the Interaction Network nodes form both the buildings and neighbourhoods. A Interaction Network node is considered a neighbourhood when it contains nested edges or nodes, otherwise it is a single building.

Each of the visual elements has its own dimensions with which the different aspects can be visualized. A full list of each visual element dimension and the mapping to the dimension of the input data is shown in Table 6.1.

A street has two visualization dimensions: width and colour. The positions of buildings and neighbourhoods, and thus the length of the streets, are set within the Interaction Map visualization. The street its width dimension, similar to a real street with cars, is scaled with the number of transitions (i.e. cars) between the two elements. The colour denotes the maximum duration of these transitions.

The neighbourhoods do not have many visual dimensions, as their size depends on the sizes and positions of the elements within. The colour, however, is influenced by the depth of the neighbourhood's level, i.e., when a neighbourhood is within another neighbourhood, it has a different colour than the parent. This allows the user to visually distinct the different levels.

The buildings, contrary to the neighbourhood and street, have an additional height axis in the 3D visualization, which can be used as a visualization dimension. In our metaphor the height is used to visualize the number of unique cases within the connected transitions. The colour of

the building depends on the maximum duration of the looping interactions/edges. Although the buildings can have a varying values for the two horizontal axes, width and length, we opt to keep these equal for all buildings, as this simplifies the understanding and interpretation of the visualization [2].

Each of the dimension input values may have values outside the range of the visualization dimensions. Hence, these values need to be scaled to the correct ranges. Since the ranges of the input values may vary per context and system, the scaling needs to be adaptive.

Chapter 7 Implementation

Chapter 6 specifies the design of InteractionCity. In this chapter an implemented tool to create InteractionCity visualizations is presented.

7.1 Implementation Framework

For the implementation of InteractionCity, ProM was chosen as the main framework. ProM, short for Process Mining framework, is an Open Source framework for process mining algorithms. It provides a platform to users and developers of the process mining algorithms that is easy to use and easy to extend [22]. It allows for creating custom plugins and features many existing plugins, such as multiple options for reading and writing XES and OCEL files. These plugins are distributed to and used by other researchers thought ProM's own custom package manager and repository. The main framework and plugins in ProM are written in Java, a cross-platform and mature programming language. Since ProM is already used by researchers in the field, InteractionCity can be used within the existing workflow without requiring new environments.

Along with a UI framework and existing plugins, ProM uses the XLog library throughout its plugins and tools to handle XES data. It provides various interfaces and classes in order to iterate over the data. A log object contains the traces, which in turn contain events. Each of these have attributes which contain the actual information.

In order to provide a full suite of tools to create an InteractionCity, various plugins are created. The following sections explain and elaborates on both the libraries and plugins mainly used for InteractionCity and the created libraries and plugins.

7.1.1 TraceTable

TraceTable is a new library as an alternative for XLog objects. An XLog object can contain arbitrary attributes per event and trace, which limits the performance when iterating over the

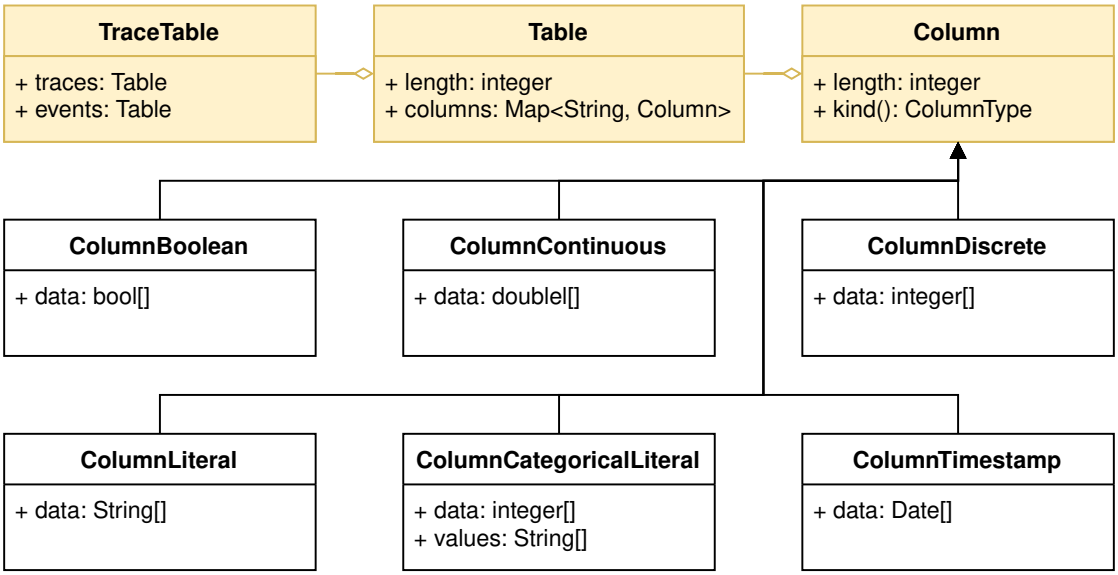


Figure 7.1: A metamodel of a TraceTable.

attributes.

The TraceTable is limited in this functionality, by requiring each trace and each event to contain the same attributes as the other traces and events respectively. Any additional attributes are dropped during the transformation from XLog to TraceTable. By setting this requirement, the TraceTable can be implemented using simple arrays for each column of attributes. These arrays provide a more performant method of handling and sequencing the attributes compared to the various object pointers required in the XLog implementation. A simplified meta-model is shown in Figure 7.1.

An additional plugin to obtain a TraceTable from an XLog was added, which does not require any further arguments, other than the original XLog, and returns the created TraceTable.

7.1.2 XLogModifier and TraceTableModifier

XLog and TraceTable provide tools for handling XES data. However, functionality to modify these is not provided. In order to provide the basic transforming of Process Mining data functionality, two new plugins are created: XLogModifier and TraceTableModifier. The transformations provided by these two plugins are equal, and hence will be explained only once. In the case of XLog, the values are stored in Attributes, which is done in TraceTable through the use of Columns. In the following paragraphs, the term attributes will be used, but can be swapped with columns, in the case of transforming TraceTables.

All plugins modify the log in-place, meaning no new log object is returned, but the given log is modified instead. In doing so, space requirements of the system do not need to be able to accommodate a, possibly, full copy of the data currently present in the log.

Renaming Attributes Each attribute is stored with its own key, a string denoting a unique name. The renaming plugin changes these keys from one to another. The user is shown a full list of the current attribute keys in the log for both the log itself, the trace, and the events, which

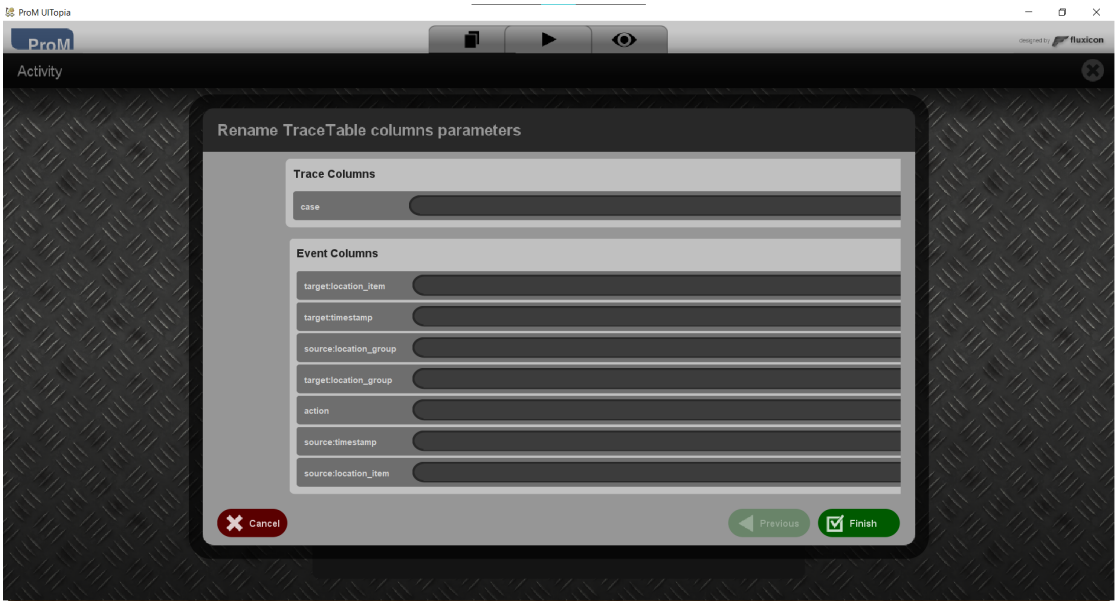


Figure 7.2: A screenshot of the Modifier plugin's rename parameter selection.

can be changed to the desired key. A screenshot depicting the selection of names is shown in Figure 7.2.

Retaining Attributes The retaining plugin presents the user with all available attributes of the log, traces, and events along with a checkbox. Depending on the selection of the user, the plugin removes any unselected attributes from the data, leaving only the required subset. A screenshot depicting the selection of attributes is shown in Figure 7.3

Retyping Attributes The retyping plugin displays the attributes of the log, traces, and events along with a selection box of the possible types of the column. The user can select the desired type of each column. The plugin then tries to transform the attribute type into the desired type, e.g. discrete to literal by printing it to a string or literal to continuous by parsing the given strings. In cases where a transformation is known to be impossible, the option will not be presented in the selection menu. A screenshot depicting the selection of attribute type is shown in Figure 7.4

Sorting Events and Traces The sorting plugin allows the user to sort the list of traces and events. The user can select multiple attributes, each of these attributes will be used with decreasing priority, i.e. only when the first attribute is equal the second is evaluated. A screenshot depicting the selection of attribute type is shown in Figure 7.5.

Flattening Trace Attributes It's often desired to keep the duplicated information to a minimum, hence some traces contain information regarding all that trace's events. An example of such data would be for which user the task is executed. Some algorithms, such as the creation of an interaction log from an event log, require the information to be placed on the events itself. This plugin removes the attribute from the traces and places them on each of the events of the trace. A screenshot depicting the selection of attribute type is shown in Figure 7.6.

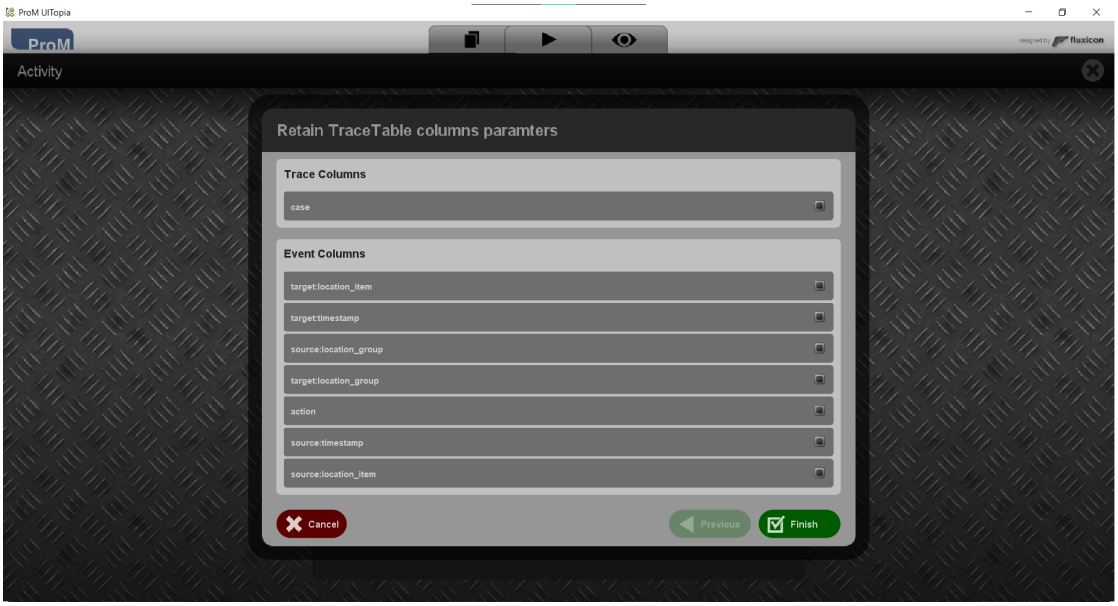


Figure 7.3: A screenshot of the Modifier plugin's retain parameter selection.

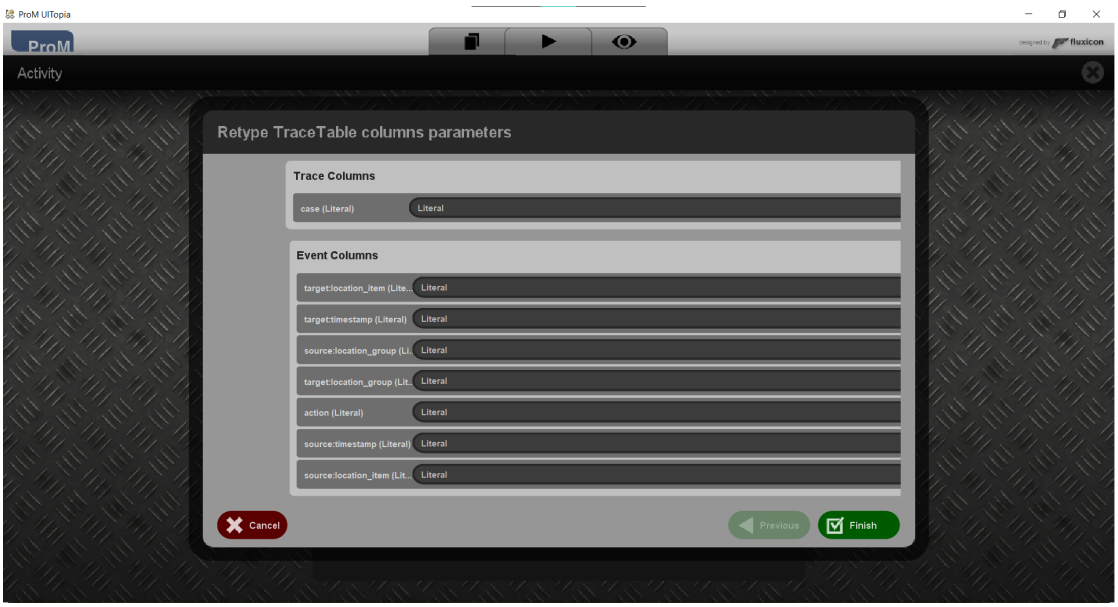


Figure 7.4: A screenshot of the Modifier plugin's retype parameter selection.

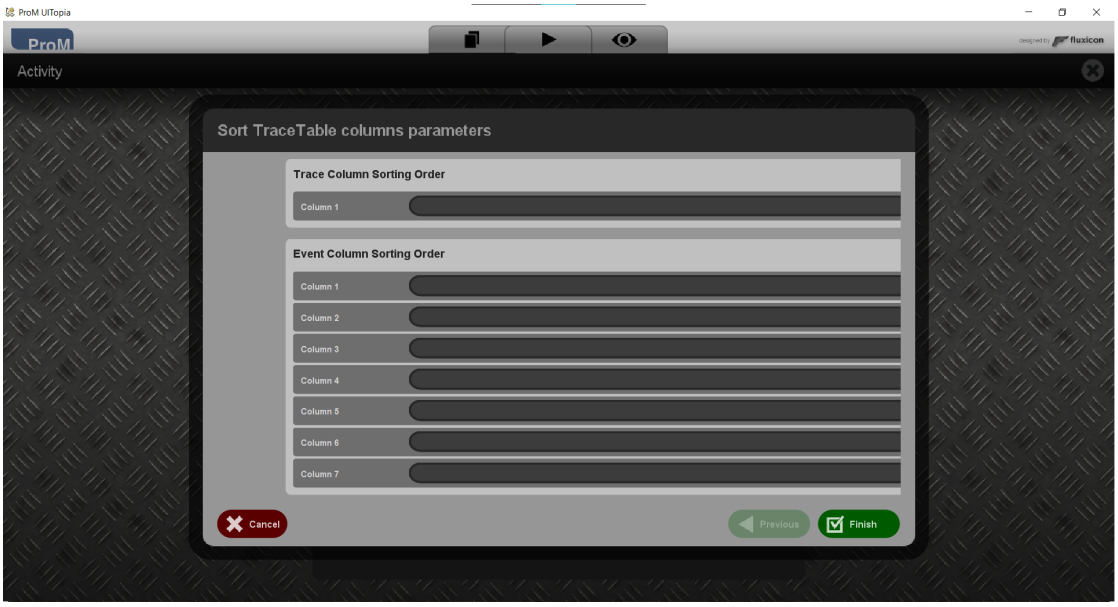


Figure 7.5: A screenshot of the Modifier plugin's sort parameter selection.

Figure 7.6: A screenshot of the Modifier plugin's move parameter selection.

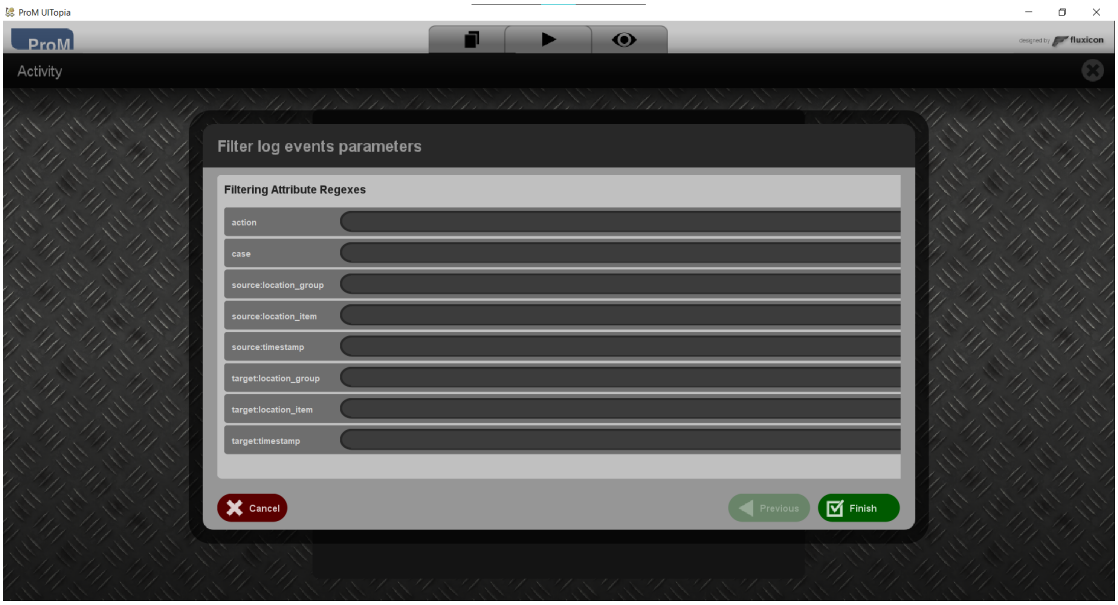


Figure 7.7: A screenshot of the Modifier plugin's filter parameter selection.

Filtering Events In some cases, events that contain a certain attribute need to be removed from the dataset. In order to do so, the filtering plugin was added. For each attribute, a Regular Expression (RegEx) can be added, which will match the on the string representation of the attribute's value. When a match is made on any of the attributes, the event will be removed. A screenshot depicting the input of RegExes is shown in Figure 7.7.

7.1.3 Interaction

Although an Interaction Log is a specification on top of the existing Process Mining logs and data, an implementation is made to enforce and statically type the requirements of the Interactions and Interaction log.

The plugin contains four main elements: `Interaction` class, `InteractionLog` interface, `InteractionLogParameters`, and three classes implementing the InteractionLog interface for Interaction Iterables, TraceTable, and XLog.

The `Interaction` class contains the required data for each interaction, i.e. case object, duration, source, and target. The `InteractionLog` interface is an iterator of `Interaction`s. In order to obtain an instance of one of the IteractionLog implementing classes, the user is prompted to select the attributes for each of the abstraction levels. These attributes are stored in a Parameters object, which can be serialized for later re-usage. A screenshot depicting the selection of abstraction level and attributes is shown in Figure 7.8.

When using the plugin, not all attributes need to be used, as a log can contain additional data. Which columns denotes which part of the FNQ

A supporting plugin is added, transition, for both the `TraceTable` and the `XLog` class. This plugin performs the task to obtain an interaction log from an event log as previously described in Section 6.2.4. This plugin does not do any additional validation on the data, requiring the user to provide the fully pre-processed data instead.

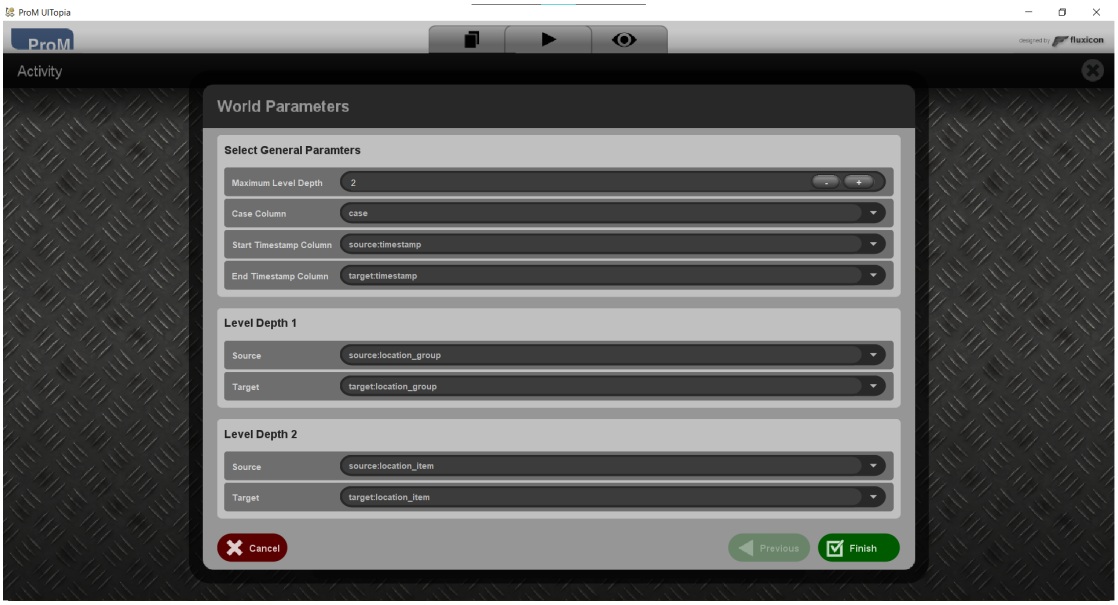


Figure 7.8: A screenshot of the Interaction plugin's parameter selection.

7.2 InteractionCity

The process to derive an InteractionCity visualization is split into several sub- steps. These steps themselves are split into separate plugins which further refine the data, in order to improve modularity.

7.2.1 Interaction Network

The first step is to create a Interaction Network. The input to obtain an Interaction Network consist of a single Interaction Log, which can be obtained with the additional plugins. The entire model can be (de)serialized, and thus stored to disk. This way, the model creation plugin only needs to be run once per dataset, and loaded again at a later time. Since the model aggregates the input data, the total size of a single model is much smaller than the raw input dataset.

The model consists of two elements; areas and streets. The streets form the interactions between areas (i.e. objects in the input data). These streets are similar to the individual interactions in the interaction log, but only contain the data in aggregated form, an Aggregated Interaction. The aggregated interactions contain four elements; the number of interactions, the number of unique cases, the minimum duration of the interactions and the maximum duration of the interactions. As the interactions between areas are bidirectional, as described in the design (Chapter 6), each street is implemented as a bidirectional connection, with some containing no data for a direction. To make the visualization more consistent, the bidirectional connections are considered "forward" for the largest direction.

The areas form the fully qualified name hierarchies, whereas the streets form the interaction between the areas. The areas form a simple tree structure, where each area contains a key, some values, and possibly children. Each key is a string being a single element of the FQN. To re-obtain the FQN of a single node, the tree is traversed upwards to the root and keys are stored into a list. The data within the areas also need to be aggregated, hence each area contains three separate

Aggregated Interactions: Incoming, Outgoing, and Looping/Self interactions.

Transforming an interaction log into a model is performed in separate steps. The first step iterates over each of the interactions in the interaction log and creates a source and target area if it did not exist in the hierarchy yet. Following, the street with that source and target area is found or created and the interaction is added to the forward direction of the street. Once these initial areas and streets are made, the street with an equal source and target are removed and added to the data of the belonging area.

With these streets removed, the grouping step is performed. In the grouping step, all interactions which do not belong to the same group, i.e. do not have the same parent, are removed and the data is added to the street between the source's and target's parent. Since all area paths start from the root area, the highest a grouping can go is the root area. The grouping is done until no street can still be grouped.

In the previous steps, each street was considered non-bi-direction, as it eases the implementation. However, these connections need to be bidirectional and hence are merged. For each pair cf streets with an equal source and target, in which one is flipped, the merging removes the smallest street and add its data to the backward direction data.

7.2.2 Map

To visualize the Interaction Network, the model is first transformed into a Interaction Map. This map is used to filter for only the required data and to position the areas and streets.

In the map there are areas (i.e. nodes) and streets (i.e. edges), these can be transformed into the existing 2D Graph implementation, provided through ProM. This graph implementation allows us to give the user an initial view of the data, although missing some key aspects compared to the desired visualization, such as depth.

Before the map is displayed, the nodes and edges of the map need to have their attributes set, such as their width and colour. Table 6.1 describes the mapping with which the map is visualized. The varying attributes are stored in an `Analogy` classes, in which the actual translation from aggregated values to visual dimensions is done.

With the map nodes and edges properly sized, they are positioned using a simple layout algorithm provided through ProM. This layout algorithm puts the elements within a single group close to another, whilst placing the top-level groups in a circle. The positions, once shown to the user, can be manually changed by the user. This allows the user to move the elements that are of interest to a better location.

The map contains some changeable parameters. In Figure 7.9, on the right side, the menu in which the visualization parameters can be configured can be seen. The first parameter, "Ignore Names", accepts a Regular Expression (RegEx) which tests each of the nodes' name and omits said node when the RegEx matches. Each of the streets has a size, denoting the number of interactions was present between the two areas. Although such streets will not be present in the input Interaction Network, this option enforces that such streets are omitted entirely.

The "Top N Edges, Depth N" parameter denotes the number of streets shown for each depth level. When the value `-1` is used, all streets are shown. Otherwise, the list of streets is sorted by size and only the largest N are used. Following the omitting of streets, the "Include All Nodes" parameter can be set. When unchecked, each of the areas with no connected streets (i.e. streets that are not omitted and are either a source or target to this area) is omitted. When checked, all areas are shown, regardless of connected streets.

The final visualized map for our running example is shown in Figure 7.9. In this visualization the default parameters, which are the parameters used in Figure 7.9 on the right-hand side, are used.

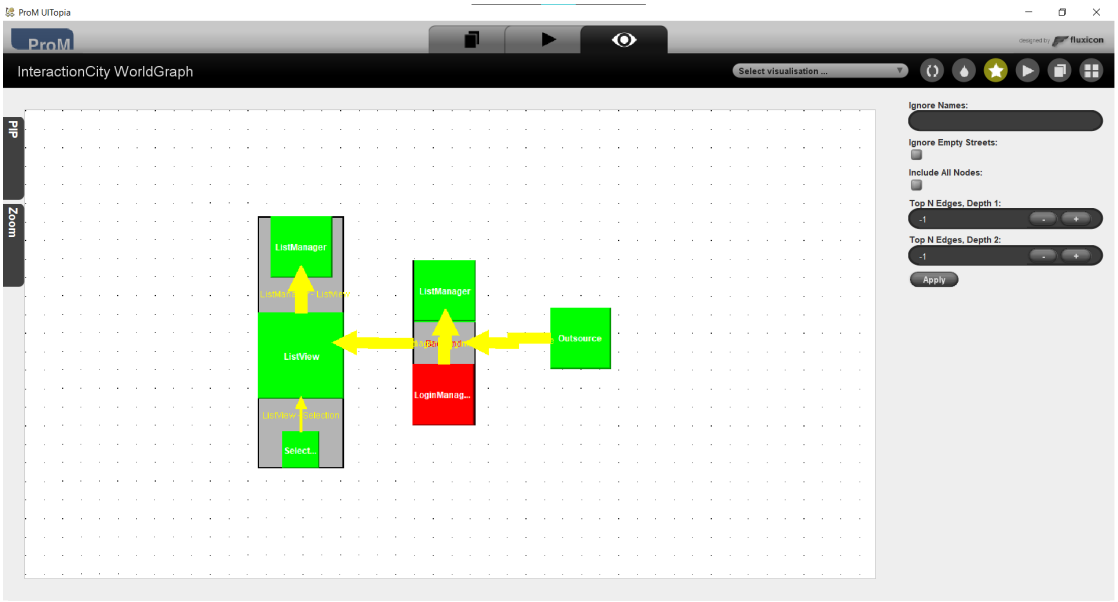


Figure 7.9: A screenshot of the Map plugin's visualization.

7.2.3 World

With the analogies created and the positions set, the final Interaction World visualization can be made. In contrary to the map visualization, this step does not have any configurable parameters. The map created in the previous step is only transformed to the new data structure, without modifying the values itself.

The final visualization is made with OpenGL, as it integrates well with ProM. The final visualization is shown in Figure 7.10. Due to time constraints, in this visualization, the options for interactivity are limited to only zooming. Further interaction methods might increase the ability for the user to obtain knowledge, which is a good topic for further research.

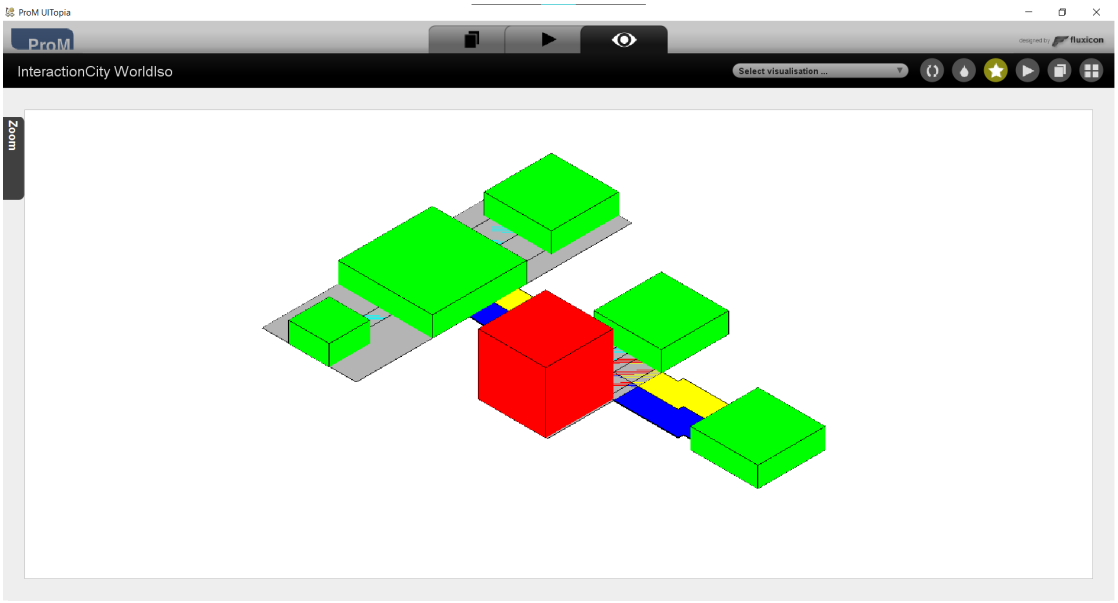


Figure 7.10: A screenshot of the World plugin's visualization.

Part IV
Evaluation

Chapter 8 Case Study

In the previous chapters we have seen both the design and implementation of InteractionCity. In this chapter we will apply the InteractionCity visualization technique to a case study.

8.1 SendIt

In this case study, we will look at SendIt, which is a postal company within the Netherlands. This dataset is the same, anonymized, dataset as used in the case study of [19]. SendIt has depots located near hotspots in the Netherlands. Within these depots, parcels can undergo different steps of the parcel delivery process. Each of these steps, i.e. activity, is recorded with an entry in the log of that depot. Each time a parcel advances a step in the process, a worker scans the parcel's `barcode`. This `barcode` and worker's activity automatically creates an entry in the log.

The entries are obtained as a single XES file. Each trace belongs to a single parcel, denoted by its `barcode`. A `barcode` is a unique identifier in the form of a string, e.g. `1ABCD2345678`. To anonymize these values, each `barcode` is hashed using the sha256 algorithm and the first 14 characters are used instead, which results in no overlapping barcodes, i.e. each barcode is unique.

Each event records multiple values; `date_time`, `location_code`, and `observation`. A summary of the values in the dataset is shown in Table 8.1. `date_time` denotes the timestamp when the scan was done, `location_code` is the depot in which the step was performed, and `observation` denotes the step that was performed. The full dataset contains more than 18 unique locations codes, however some of these are not located within the Netherlands or were not parsable. Hence, these locations are mapped to a single "Other" depot. To anonymize the depots these are numbered and the old name is dropped. The "Other" depot is always mapped to the number `0`.

The `observation` consists of two parts: the type and the reason. The type is a single letter, acting as an identifier into which category the observation belongs. An example of such category is type `I`, which means that a parcel is delivered. The reason is a two-digit number, such as

Key	Type	Example Value	Description
barcode	String	7c8c45e6644b6d	The unique hashed barcode of each parcel.
date_time	Date	2016-02-18T 10:37:04	Timestamp when the scan was performed.
location_code	Integer	0	The depot the event was recorded, mapped to a number.
observation	String	J05	The step performed, composed of a type and a reason.
observation_type	String	I	The letter part of the observation, which is a category, grouping multiple activities together.
observation_reason	Integer	05	The number part of the observation, a concrete activity within a category.

Table 8.1: Values available in the SendIt dataset.

Observation	Description
A01	Shipment reported
A04	Pre-notification generated by PostNL
A16	Client matched as receiver of parcel
B01	Shipment accepted and in sorting process
I01	Parcel is delivered to the client
I10	Parcel is delivered at neighbours of client
J01	Parcel is sorted
J08	Second delivery. Parcel could not be delivered, will be tried again tomorrow
J10	Parcel received at sorting centre
J11	Parcel is delivered to retailer
J30	Parcel collected by 'planbalie'
J04	Parcel is loaded onto truck
J40	Parcel is loaded onto truck
J05	Shipment is out for delivery

Table 8.2: Observation types in the SendIt dataset.

reason 05, and denotes a specific version of the activity category. This meaning of the actual value depends on the category context, an example is reason 10 within type I, meaning delivered at neighbours.

The dataset contains only the observations with known meanings, as originally listed by [19] and shown in Table 8.2. Additionally, traces not ending in an type I observation, i.e. delivered, are removed, as these are not yet completed. Although [19] lists more observation types, any observation for administrative purposes are excluded (type A04 and type A16, both shown in Table 8.2). The dataset contains 168058 parcels and 1113720 entries.

The knowledge of the system is currently lacking; only information on the elements within the dataset is provided as well as general knowledge on postal services. Any further knowledge about the system is missing. Hence, we obtain such knowledge by using the InteractionCity technique. The following sections will follow the Visual Analytics steps as depicted in Figure 3.1.

Old Key	New Key
barcode	case
date_time	timestamp
location_code	fqn-1
observation	
observation_type	fqn-2
observation_reason	fqn-3

Table 8.3: Renaming mapping in the SendIt dataset.

8.2 Interaction Log and Interaction Network

As a first step, an Interaction Log must be created for the given dataset. It already contains the raw values needed to meet the requirements, but pre-processing is required.

The input data is in the form of an XES file containing a large amount of events, hence it would be faster to transform the dataset to an TraceTable, however to provide better readable snapshots, we opt to keep it in the XES format.

Two approaches can be taken when creating the interaction log, either the transition plugin (Section 7.1.3) can be used before or after other pre-processing steps are taken. In this case study, the transition plugin will be used last to keep the dataset during the in-between steps smaller.

As listed previously, an interaction log requires us to have a case object, start and end timestamp, and a source and target FQN identifiers. The case identifier and timestamps will be given through the `barcode` and `date_time` attributes respectively.

The source and target identifiers will be split across multiple attributes, each denoting a single part of the FQN. The raw dataset is combined from split event logs, hence the depot, i.e. `location_code` attribute, in which the event was recorded, will be the first part of the FQN. Following, the split `observation`, i.e. `observation_reason` and `observation_type`, itself is used as the following parts. Since the `observation_type` has little meaning without `observation_reason`, it will be used as the lower level.

Since the `observation` is already used through its decomposed parts, this attribute can be removed. Lastly, to make attributes easier to understand, we rename the attributes according to the mapping shown in Table 8.3. The `observation_reason` attribute is currently an integer, but will be part of the FQN, and hence needs to be changed to a string attribute. Lastly, the log is sorted. The sorting order will be done on timestamp only, i.e. `date_time`. This preserves the internal order of events, in case two events happen at the same time. The order of the traces themselves is irrelevant for the interaction log.

With this initial pre-processing applied, a small portion of the dataset is shown in Example 8.1. This pre-processing is done with the plugins described in Section 7.1.2.

Before the dataset can be transformed to an interaction log, two more plugins need to be executed: the transition plugin to obtain the interactions between events and the trace flattening modifier plugin, to move the case object attribute to the events itself rather than the traces. A small portion of the dataset after these steps is shown in Example 8.2.

With the fully pre-processed, it can be transformed into an Interaction Log. The parameters for this transformation are relatively straight forward, due to the renaming of the attributes done earlier, and are shown in Table 8.4. This Interaction Log can then be used as the input for the creation of a InteractionCity World, which does not require any additional parameters.

```
...
<trace>
  <string key="case" value="7c8c45e6644b6d"/>
  <event>
    <date key="timestamp" value="2016-02-18T10:12:06+01:00"/>
    <string key="fqn-2" value="B"/>
    <string key="fqn-3" value="1"/>
    <string key="fqn-1" value="10"/>
  </event>
  <event>
    <date key="timestamp" value="2016-02-18T10:12:07+01:00"/>
    <string key="fqn-2" value="J"/>
    <string key="fqn-3" value="40"/>
    <string key="fqn-1" value="10"/>
  </event>
  ...
</trace>
...
```

Example 8.1: The dataset after retaining and renaming attributes.

Key	Value
Depth	`3`
Case	`case`
Start Timestamp	`source:timestamp`
End Timestamp	`target:timestamp`
Depth 1 Source	`source:fqn-1`
Depth 1 Target	`target:fqn-1`
Depth 2 Source	`source:fqn-2`
Depth 2 Target	`target:fqn-2`
Depth 3 Source	`source:fqn-3`
Depth 3 Target	`target:fqn-3`

Table 8.4: Interaction log parameters for the SendIt dataset shown in Example 8.2.

```
...
<trace>
  <event>
    <string key="case" value="7c8c45e6644b6d"/>
    <date key="source:timestamp" value="2016-02-18T10:12:06+01:00"/>
    <string key="source:fqn-1" value="10"/>
    <string key="source:fqn-2" value="B"/>
    <string key="source:fqn-3" value="1"/>
    <date key="target:timestamp" value="2016-02-18T10:12:07+01:00"/>
    <string key="target:fqn-1" value="10"/>
    <string key="target:fqn-2" value="J"/>
    <string key="target:fqn-3" value="40"/>
  </event>
  <event>
    <string key="case" value="7c8c45e6644b6d"/>
    <date key="source:timestamp" value="2016-02-18T10:12:07+01:00"/>
    <string key="source:fqn-1" value="10"/>
    <string key="source:fqn-2" value="J"/>
    <string key="source:fqn-3" value="40"/>
    <date key="target:timestamp" value="2016-02-18T10:37:04+01:00"/>
    <string key="target:fqn-1" value="10"/>
    <string key="target:fqn-2" value="J"/>
    <string key="target:fqn-3" value="5"/>
  </event>
  ...
</trace>
...
```

Example 8.2: The dataset after applying the transition and flattening plugin.

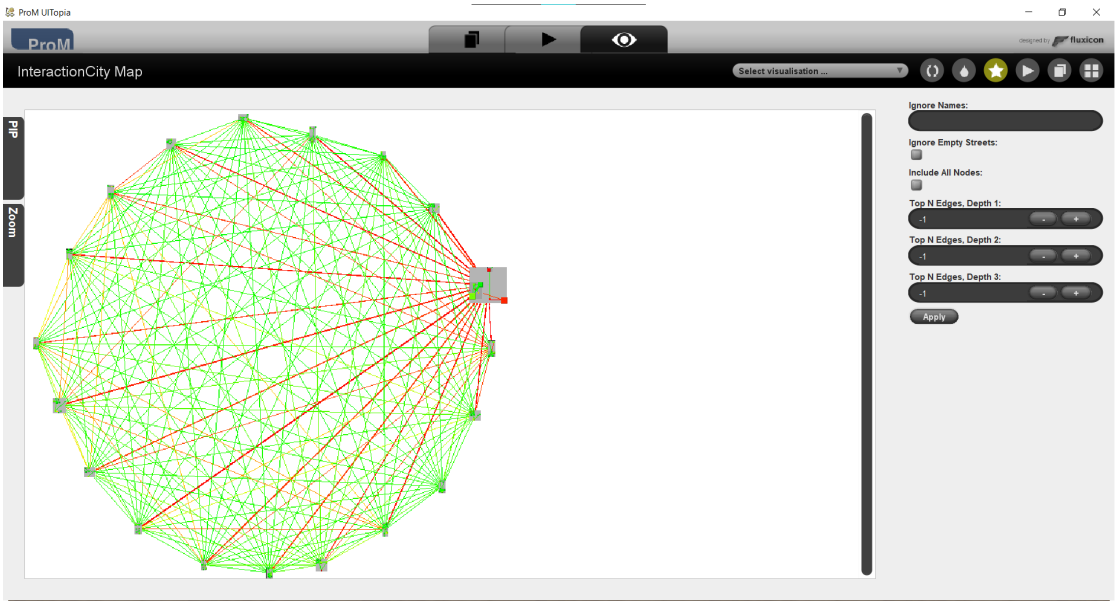


Figure 8.1: The Interaction Map using default positioning and the initial parameters.

8.3 Interaction Map

With the Interaction Network, a Interaction Map is created. Upon starting the plugin, the default parameters and layout are used. This initial graph is shown in Figure 8.1.

In this visualization, we see a single area connected to a large amount of the red streets. The name of this area is `0`, which we have previously seen in Section 8.1 as the "Other" depot. This depot does not provide much information on factors within our system, hence remove it from our dataset. This step is performed through the Filter Plugin, described in Section 7.1.2. Since this plugin cannot be used on a city, the filtering will be done on the input data and the model will be re-mined. An alternative is to add the depot to the "Ignore Names" parameters. However, this only hides the depot from view, rather than remove it from the model, and hence the outgoing parcels of "Other" will still be counted in the incoming values of the other depots. The Interaction Map of Figure 8.4 with the new parameters is shown in Figure 8.2.

This visualization contains a large amount of areas and streets, which causes too much noise to extract context knowledge from. Hence, we will apply filtering through the provided parameters. Firstly, the checkboxes for "Remove Empty Streets" and "Include All Nodes" are checked. Next, the streets will be limited per abstraction level. The total number of streets per level are: 171, 63, and 200. When we remove too many streets, some areas will become disconnected. For our map, all top-level areas remain connected when using any value above 131, as shown in Figure 8.3. For our case study, this leaves too much noise, hence we opted to limit the number of disconnected top-level areas to 3 and use the minimal number of edges meeting that requirement; 23. The streets on lower levels should also be reduced, as they do not contribute to the majority of the parcels and cause a large amount of noise. Hence, we arbitrarily opted to select 50% and 33% of these streets, resulting in 32 and 67 streets for depth 2 and 3 respectively. Using these parameters and the default layout, the results shown in Figure 8.4 are obtained.

The default layout places the areas in a circle, but does not optimize for non- overlapping streets. Hence, the last step is to manually position the areas and streets. The result of this final

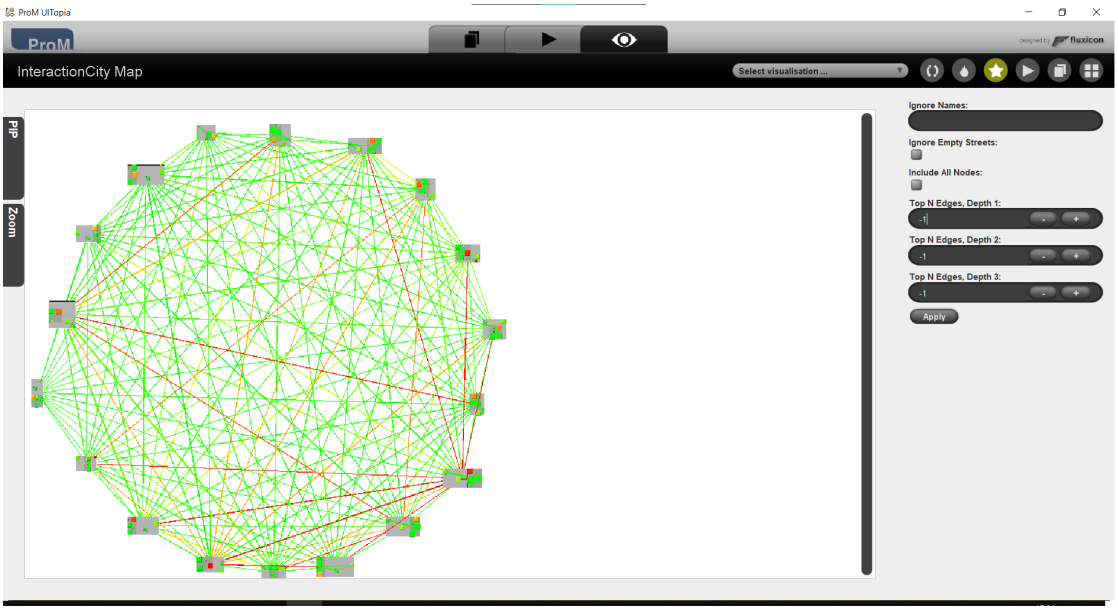


Figure 8.2: The Interaction Map using the default positioning and tuned parameters, with "Other" removed.

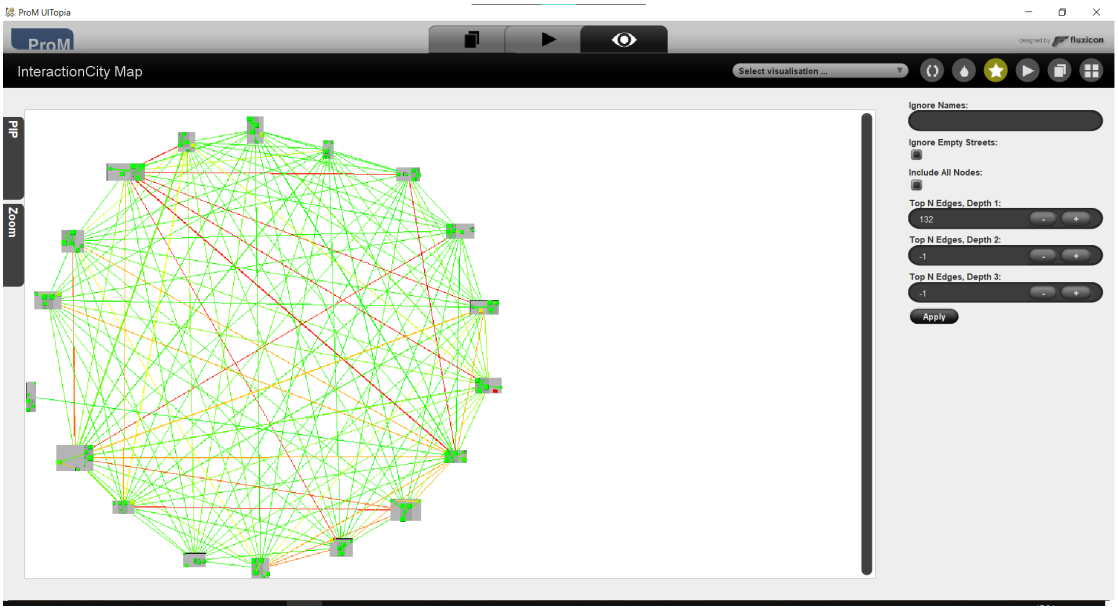


Figure 8.3: The Interaction Map using the default positioning and tuned parameters, with 132 streets on depth 1.

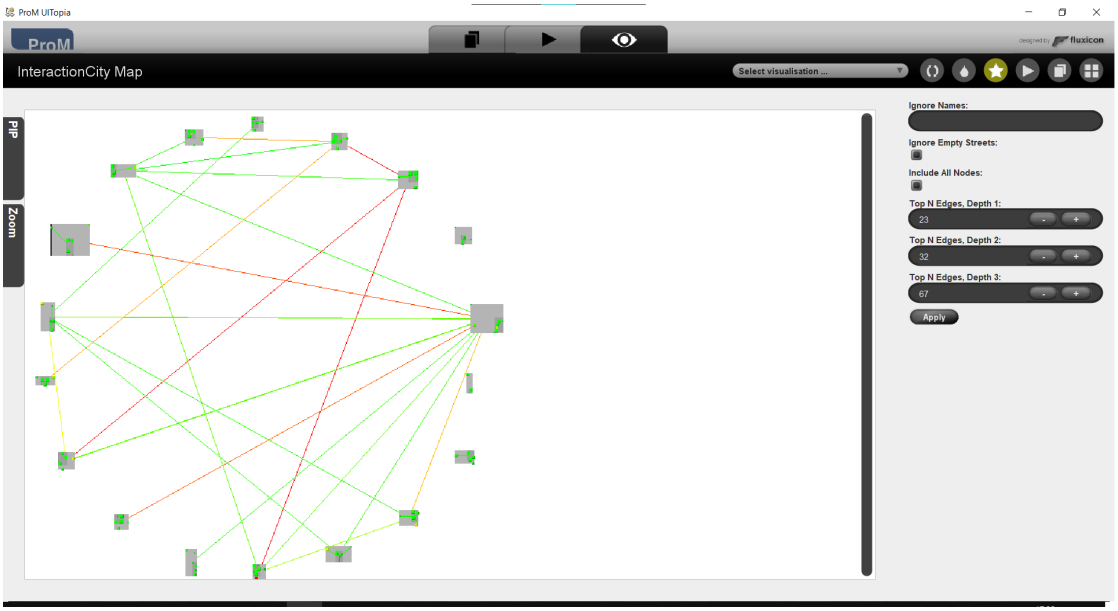


Figure 8.4: The Interaction Map using the default positioning and final parameters.

map is shown in Figure 8.5a. As the depot numbers are too small within the figure, an additional figure, with better legible depot numbers added, is shown in Figure 8.5b.

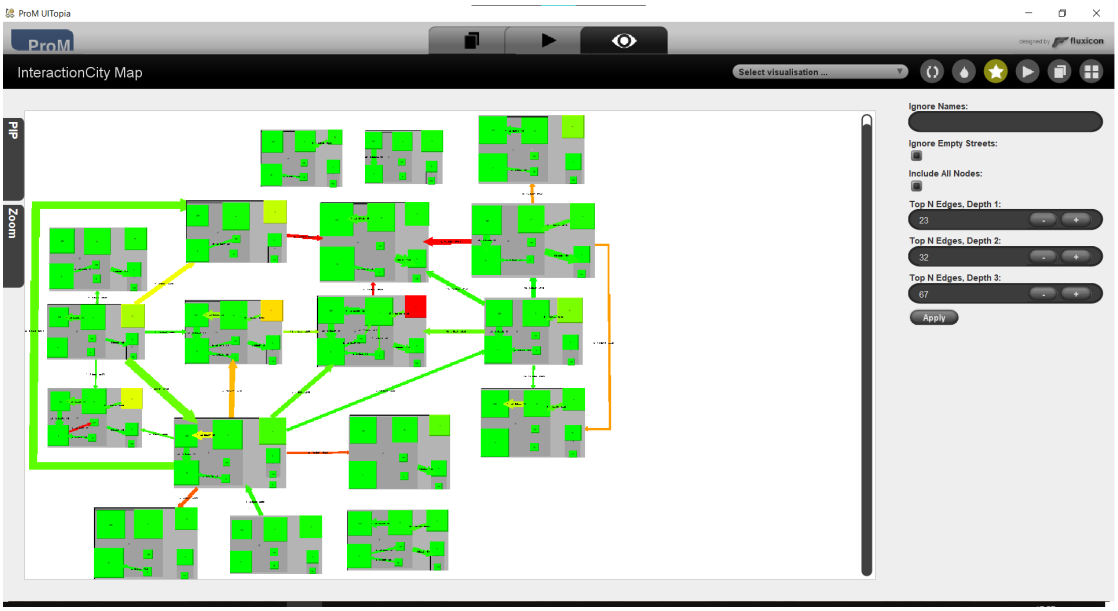
8.4 Interaction World

Using the Interaction Network and Interaction Map previously created, we can use these as the input to create an Interaction World, for which no additional parameters are needed. The resulting Interaction World is shown in Figure 8.6a, and in Figure 8.6b with the depot numbers added.

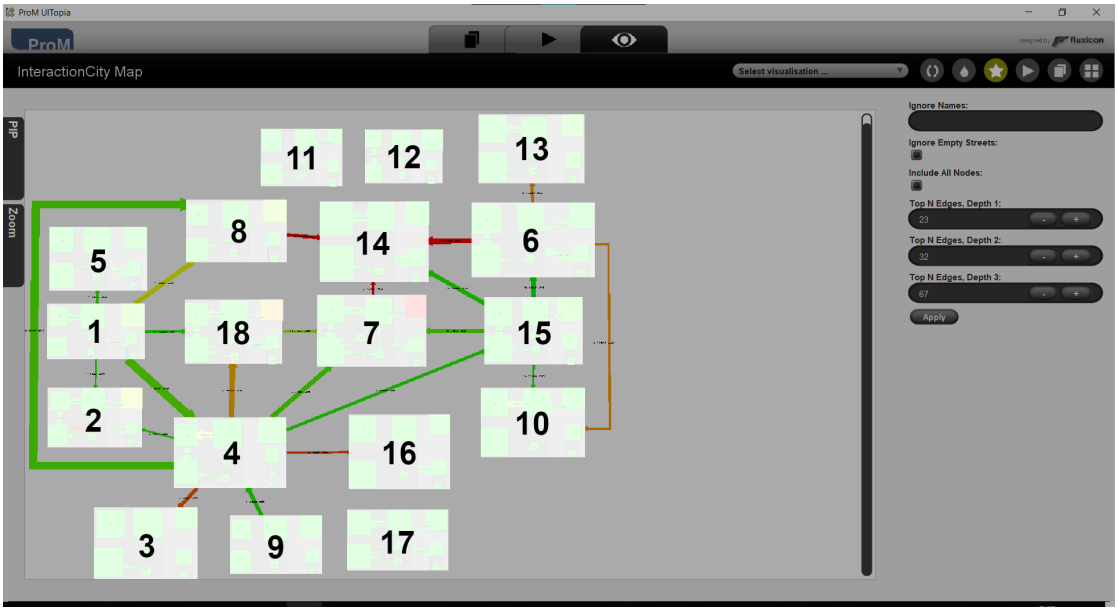
What kind of knowledge we want to obtain from the visualization, heavily depend on the needs of the user. In our case, any architectural knowledge was lacking, which results in the need for any basic knowledge of the number of buildings and the interactions between buildings. This information is provided through Figure 8.6.

Another use-case could be resolving any bottlenecks of deviations from the average process. The visualization provides the user with the elements that might have such deviations. The analogy of the visual dimensions was listed in Table 6.1. In the visualization of SendIt are some notable elements, which can then be used as new input for a further investigation on why these elements deviate. In our case, notable elements and their conclusions or follow-up questions could be:

- The initial Interaction Map shows a fully connected graph between depots. This implies the depots do not send each parcel to a selected depot prior to further shipping. This is the expected case to ensure each parcel takes the shortest path.
- Each Depot contains the same amount and type of buildings. Additionally, within the depot, the streets between the same building pair have roughly the size and ratio. This is indicates that each depot follows the same process pattern for each parcel.



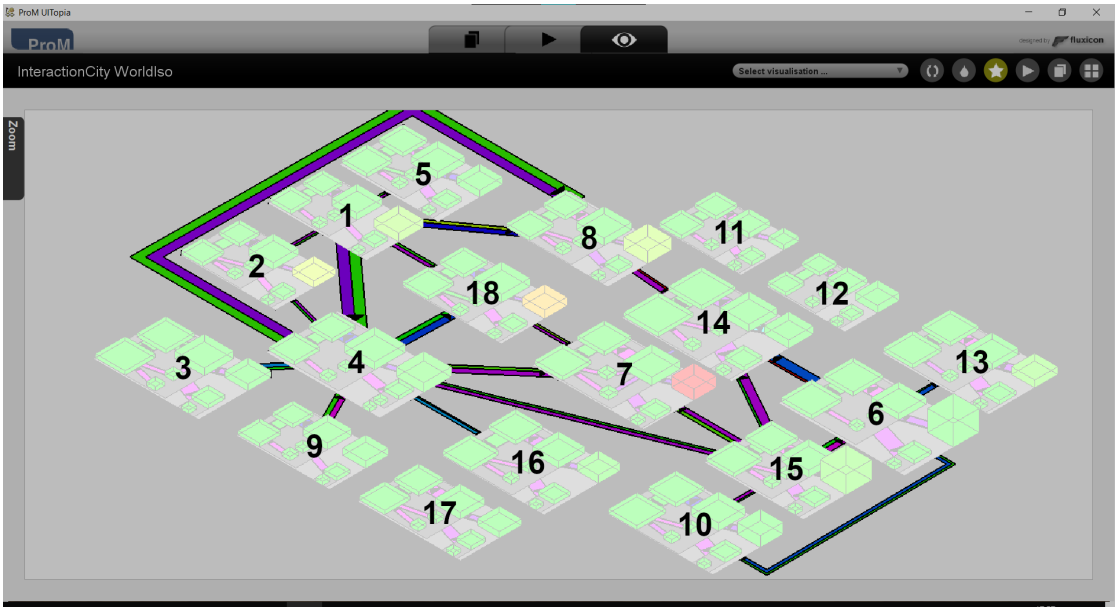
(a) The created Interaction Map visualisation.



(b) Figure 8.5a with the Depot numbers added.

Figure 8.5: The Interaction Map using the manual positioning and final parameters.

(a) The created Interaction World visualisation.



(b) Figure 8.6a with the Depot numbers added.

Figure 8.6: The Interaction World using the Interaction Map from Figure 8.5.

- Streets Within Each Depot have a very one-directional flow of parcels, taking the path type `B`, type `J`, to type `I`, with a much smaller number of parcels going back from type `I` to type `B`. Considering the meanings of the `observation_type`s, from Table 8.2, this is in line with the expected behaviour; sorting, handling, delivering, and resorting when undelivered.
- Depot `4` has an additional building, type `A`, compared to the average depot. In our set of `observation_type`s there is only one type `A` type; "Shipment reported". Why only depot `4` has this `observation`, is a good follow-up question for deeper study of the system.
- Depot `4` has more connected streets than others. This indicates the Depot is a hotspot for parcels. The height however, is not much larger than other buildings, meaning the number of unique parcels is not scaled with the amount of incoming and outgoing traffic. Thus, parcels pass through this Depot multiple times.
- Buildings `2.B.1`, `7.B.1`, and `18.B.1` have a yellow, red, and orange colour, respectively, compared to the average building (green), i.e. have a larger maximum duration of looping-transitions. This is an indication that the process within these Depots take longer than is to be expected. Why this is the case should be investigated further by the user.
- Buildings `6.B.1` and `15.B.1` are relatively higher than other buildings, i.e. have a larger number of unique parcels. Since this is the `B01 observation`, we can conclude that most of the parcels arrive at these depots.
- Streets between Depots have varied ratios. A ratio of roughly 50% means the number of parcels send and received between a depot pair is roughly equal, e.g. between depots `4` and `8`. When a ratio is deviating, as is the case for each of the streets connecting to depot `14` where the ratios are towards the 80%, this means the parcels send between the two depots is heavily favoured towards one depot.
- Streets ending in the type `I` neighbourhood have varied sizes per depot. The type `I observation_type` is the delivery of the parcel at the final destination. Hence, the size of this street is a good indication of the amount of parcels with their destination being handled by this depot. On the contrary, when the outgoing streets of the depot are large with a small type `I` street, most of the parcels are transported elsewhere.
- Various Streets have a red/blue colour compared to the average green/ purple. The colour of a street denotes the maximum duration of the transitions. Why these transitions have a longer duration is a question to be further investigated by the user.

8.5 Runtime Enterprise Architecture Visualization

In the study of [19], another visualization of the same dataset is made, Runtime Enterprise Architecure (REA), shown in Figure 8.7 and previously mentioned in Section 4.5. In their visualization, each of the depots is named "Loc 1" to 'Loc 18', with the various `observation_type`s shown as boxes above the depot numbers. Each of the depots has a width and a height dimension. The width dimension scaled with the number of parcels sent, whilst the height scales with the number of parcels it handles. Contrary to our implementation, REA uses the median duration for each of the colours rather than the, in our study used, maximum duration.

Since the visual elements are grouped on `observation_type`, a new Interaction World is made to have a similar grouping. This visualization is shown in Figure 8.8.

The most notable depots in the visualization of [19] are:

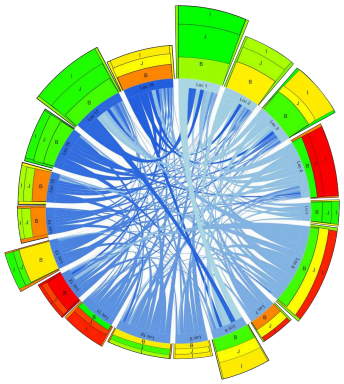


Figure 8.7: Runtime Enterprise Architecture visualization, by [19, Figure 50].

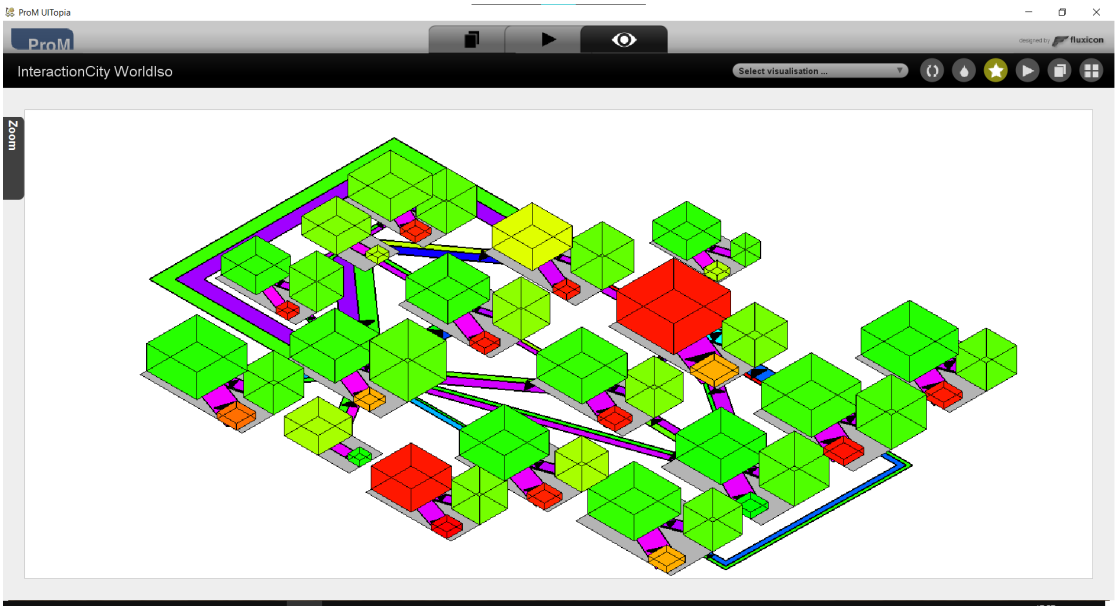


Figure 8.8: The Interaction World of SendIt grouped by `observation_type`.

- Loc 1, 8, and 13 with an above-average height, whilst having an average width.
- Loc 4, 7, 11, 12 with multiple red or orange `observation_type`s.
- Loc 6 with a much larger width, whilst having an average height.
- Connection between Loc 1 and Loc 8 which has a larger size than average connections.

8.5.1 Comparison

Comparing the InteractionCity visualization to the REA visualization, there are various differences, impacting the knowledge the user can obtain. Both visualizations use three main dimensions: width, height, and colour. The impact of the knowledge step is roughly the same. However, the by using the median duration rather than the maximum duration, REA creates a more nuanced scaled in colours. These colours also provide better information on the element as a whole, rather than the single extreme case. Although the Interaction World provides a 3D visualization, whilst REA a 2D visualization, the usable axis of both is the same, as InteractionCity uses the same sizes for both horizontal axis.

The main difference between the two visualizations, is the information provided through the connections between depots and the positioning of the depots. In the REA visualization, the connections between nodes are grouped to be on an inter-depot level only, whilst InteractionCity also provides information on a more detailed level. Furthermore, the ratio between the number of parcels in each direction of a street is not provided, instead visualized indirectly through the ratio of width and height in an aggregated form per depot. Most impactful to obtain knowledge, however, is the positioning of the depots. By placing each of the depots on a circle, the connections between depots overlap heavily. By using either a good layout algorithm, or manual placement, the spatial awareness of a user can be used more effectively. An example of which, are areas with a higher level of connectivity or (almost) circular dependencies, which is not the case for the chord diagram used in REA.

8.6 Concluding Remarks

By using the SendIt data as a case study for the InteractionCity visualization technique, we have seen how it can be used in practice. The main take-away is positive; the visualization is easy to interpret and provides a good initial, albeit coarse, step regarding the system's architecture and points of interest. We have seen that using interaction data, rather than event data, results in a visualization which does not lack in its ability to show the locations from which the events or interactions originate, instead providing a good hybrid where both are equally important. Additionally, using the various visual dimensions, such as colour and width, is an effective method to provide the user with visual indicators. However, this requires a good mapping from raw data to these visual dimensions, which was mostly achieved. Lastly, using the FQNs of the elements within the log resulted in hierarchies which were both quick to comprehend and worked well for the implementation.

Although the results of the InteractionCity visualizations are positive, the case study and comparison with REA also resulted in some aspects where it could be improved. First, we will look at the aspects of the InteractionCity design which can improved, after which improvements of the current implementation are listed. For the design, there is one aspect which requires improvement; the currently only used example mapping, shown in Table 6.1. This mapping could be investigated further, to provide the user with, possibly, a better mapping which either quickens obtaining knowledge from the visualization or guides the user to better points of interest.

Additionally, the main lack within the current mapping is the use of extremes, through the maximum duration used for the colours, not only provides information on a single interaction rather than the collection of interactions within the element, it also affects the other elements which are related regarding scaling, resulting in a less nuanced amount of colours.

For the implementation, there are various improvements that can be made to ease the use of the tool and allow the user to better find the desired knowledge. Firstly, a good layout algorithm can be found, which provides the user with a better, much less clustered and overlapping, initial positioning than the current circle layout. In doing so the task of positioning should not only become quicker, but also simpler. The ability to move elements as the user sees fit, however, should remain in the Interaction Map visualization. The map visualization could also provide more options for the user to modify the visualization, such as through better filtering. Furthermore, the mapping with which the visual dimensions are created, should be modifiable according to the needs and wants of the user. This way, each visualization could provide the user with a different kind of information, even though the input Interaction Network stays the same and all the data does not need to re-mined. Closely related to mapping, is the scaling of the values. By allowing the user to both change the range and method, such as linear or logarithmic, an emphasis can be placed on the smaller or larger values. Lastly, a smaller change is the visualization of the ratios within the Interaction Map, as these can inform the user for a better choice regarding the filtering, and possibly scaling, of the elements.

The Interaction World visualization also poses various points for improvement. A low-hanging fruit for improvement of this visualization, is the addition of displaying more information on the visual elements, such as the names of elements. Furthermore, the world should provide better interaction. This can be done by allowing the user to interact with the world, such as by viewing the raw values of an element on demand or providing a method to visualize a single trace through the world.

During the case study, we found the current implementation, where the user can only zoom towards the centre, lacking when details on elements on the edge of the map are desired. Instead, we would rather see an implementation which steps away from the current isometric visualization, but instead provide the user with a free camera, with which the user can fly through the city. In doing so, the city metaphor might also be more intuitive. Another, somewhat related, improvement would be to ease the transition from a Interaction Map to a Interaction World and back. In the current implementation, the user uses the first as an argument of a plugin to create the latter. Although currently not a major issue, this would increase the workflow when creating the visualizations.

When a free camera functionality will be added or the Map and World visualization are going to be merged, an alternative framework might prove more fruitful. Although providing the user with much built-in functionality, a framework with a larger focus on the ability to view the world in a 3D setting can increase the ease of use and better match the expectations of the user regarding the functionality of the visualization.

Chapter 9 Conclusions

At the start of this study, a design objective was defined, Chapter 1, and research questions were formulated, Section 2.1. Following from this objective, a new visualization technique, InteractionCity, was proposed and implemented. In the following sections, we will look back on the questions and opportunities for future work discussed. To start, we will repeat the design objective:

Improve the architectural and context knowledge *by* creating a new visualization technique *that* uses the system's execution data *in order to* improve the understanding of said system.

9.1 Answers to Research Questions

At the start of this study, a Main Research Question was asked, from which supporting sub-questions were formulated. Hence, prior to answering the Main Research Question, the sub-questions will be answered first.

9.1.1 Sub-Question 1

Sub-question 1 is formulated as follows: *How can we capture and represent dynamic information about the usage of software systems efficiently?*

As stated in the question, the main focus is on dynamic information, rather than any static information. From the seen background literature, only Process Mining proposes two formats to store such dynamic data; XES and OCEL. Both of these formats focus on events occurring within the system. XES, compared to OCEL, is more limited in its uses, as it requires each event to

belong to a single trace and cannot have additional objects linked to each event. However, XES is currently better supported than OCEL. In this study, we defined another format which focuses on the interactions between elements rather than single events within a trace; an Interaction Log. This format allows for different kinds of objects to be linked with another through *iteractions*. The dynamic information itself is stored within such interactions, whilst the objects it connects to are implicit.

In this study, we have seen how an XES event log can be transformed into an Interaction Log, through various simple transformations of the data. How the dynamic data is captured and stored in any of the formats, can be achieved through logging within a system. What information is logged and the actual implementation of the logging, is, however, system dependent.

9.1.2 Sub-Question 2

Sub-question 2 is formulated as follows: *What is the current state of the art in visualizing static and dynamic information for software architecture?*

In Part II, three main background concepts are introduced; Visual Analytics, Software Architecture, and Process Mining. Each of these have their own fine-tuned methods and techniques that meet certain needs within their context.

In Visual Analytics, the focus mostly lies upon hierarchical information visualization, forming trees and nested structures. Nonetheless, Visual Analytics does provide a framework which is used as basis for the knowledge obtaining process in this study. The techniques proposed in Visual Analytics can be applied to both static and dynamic information.

Software Architecture focuses on the reconstruction of an existing system, which in turn provide the input for the visualization techniques. These visualization techniques can be categorized in three types; Graph-, Matrix-, and Metaphor-Based Techniques. Out of which, the Metaphor-based techniques are the least constrained in their visual representation. Although Software Architecture often uses both static and dynamic data on the system to create its visualizations, it mainly focuses on the static aspects of the system within this data. Example of such aspects are the various classes within the source code. Hence, the knowledge Software Architecture provides on the system is its components and their relationships, rather than their behaviour during execution.

Lastly, Process Mining uses event logs, i.e. dynamic data, in order to obtain knowledge. The static data, such as source code, is not used at all. The visualizations created for Process Mining can be in either an aggregated or instance form. The aggregated form merges various events and connections found in the system to provide an overview over the system as a whole, whilst instance visualization allows the user to evaluate all events through one visualization, e.g. in order to find anomalies or patterns.

9.1.3 Sub-Question 3

Sub-question 3 is formulated as follows: *What visualization techniques can be developed to improve the understanding of a software system's behaviour?*

In previous research, we have seen various visualization techniques, tackling the task of obtaining knowledge from software's dynamic information. In this study, we propose another technique, InteractionCity, which iterates on these techniques and further defines the requirements for the data.

InteractionCity uses two different visualizations; a Map and World visualization. The map provides the user with the ability to focus on the positioning of visual elements rather than the details of the elements. The World visualization, on the other hand, does not allow the user to

make any further changes, but rather focuses on the details of the various elements. Both of these visualizations use a city metaphor to map the found dynamic data to a visual presentation.

This visualization technique visualizes uses the dynamic information only, but uses a hybrid method regarding static information visualization found in Software Architecture and dynamic information found in Process Mining. The static component can be found in the visualization of the various elements as a hierarchy, whilst the dynamic data is mostly found in the connections between the elements.

9.1.4 Sub-Question 4

Sub-question 4 is formulated as follows: *How is the developed visualization technique perceived and compare against existing techniques?*

We evaluated the InteractionCity visualization technique through a case study, as described in Chapter 8. Here, the full process of obtaining an Interaction Log from an event log is shown and the visualizations are made. Afterwards, in Section 8.6, the process and results are evaluated and compared to existing techniques.

For the design of the technique, the results of the evaluation were overall positive; the visualization provides an easy method to obtain an overview of the system. The visualization successfully displays both the grouping of the static elements and the interactions between these elements without introducing much clutter. Additionally, it allowed us to quickly find points of interest within the case study, which provide a good stepping stone to obtain further knowledge on the system and points where improvements could be made.

Regarding the implementation, the user cannot obtain fine-grained details on the various elements within the system, which might be a desired option in the future. This and further points of improvement were highlighted in Section 8.6.

9.1.5 Main Research Question

The Main Research Question is formulated as follows: *What are techniques to support the architect in understanding the behaviour of a software system?*

Such supporting techniques have been shown in this study, being; Visual Analytics, Software Architecture, and Process Mining. Most important for such technique is to meet the requirements set for Visual Analytics, as shown in Section 3.2, which were also seen in the evaluation through the case study in Chapter 8. Additionally, visualizing the actual behaviour of a system requires the dynamic data, rather than any static data of the system.

Although many techniques exists, most of these only focus on a few aspects. The InteractionCity visualization technique attempts to combine all of these aspects in order to achieve this task and join the strengths of other techniques.

9.2 Limitations

In this study, the InteractionCity technique is designed and implemented. Unfortunately, a created technique is never perfect and can be improved. The initial improvements on the implementation, have already been listed in Section 8.6, and as such these will not be repeated. Instead, this section will look at the other aspects that can be improved.

Firstly, in the current design and case study the initial data was assumed to be some form of event log. This resulted the pre-processing steps and transformation to be compatible with event logs. Since the used dataset was not collected within this study, an alternative approach was not

feasible. However, an alternative approach of both capturing and storing the interaction data might introduce entirely different issues, unforeseen when using XES data.

In Section 7.1, the usage of the ProM framework was chosen. However, this framework still uses older versions of Java and most of its documentation has not been updated in several years. Although the concepts of having existing packages and a package manager to distribute these packages, it was found that many of the required and desired pre-processing steps were not readily available, and instead required a custom implementation. ProM's framework does ship with a full XES read and write functionality, but used programming concepts which aren't shared by most programming languages. All of these aspects resulted in a reduced ease of development and required more time to implement than initially anticipated, resulting in a less than hoped, although successful, implementation.

Additionally, ProM uses the UI framework "UITopia" internally, which is mainly a 2D focused UI framework. This did provide a 2D graph implementation, but poor (native) support for 3D capabilities. This not only further added to the time constraints, but also limits the possible features that the InteractionCity visualization might require in a future iteration.

Lastly, by continuing on the work of [24], the initial designs of InteractionCity have been mostly city-metaphor based. It is possible that, keeping the concepts of Interaction logs with interactions rather than events, an alternative type of visualization could have been found that provides better visualizations.

9.3 Threats to Validity

Every research is subject to various threats to its validity. These threats are split in three categories: internal validity, external validity, and reliability. Internal validity is whether the results found by itself are valid and not influenced by various factors, such as certain features within the dataset. External validity denotes the extent in which the results found in this study are generalizable to other situations, such as a different dataset. The reliability denotes the reproducibility of the results within this study.

The dataset used in this study was the same as was used in the study of [19], hence the aspects regarding both internal and external validity, as described in their Chapter 11, on the dataset apply here to the evaluation case study as well.

The internal validity of this study was limited by using a prior used dataset, of which the state regarding context knowledge and the layout of the dataset is known. Furthermore, the pre-processing, i.e. filtering, steps taken on this dataset and their reasoning have been explained without regards to any features of the resulting dataset. Nonetheless, the data might have some consistencies as the data has not been fully reviewed. An example of this might be the single type `A` observation mentioned in the case study.

To increase the external validity of this study, the technique and its processing steps are designed and developed independent of the dataset used. This should allow this technique to be applied to other datasets, given that the data is provided in the correct data-format. During the development, a small subset of the dataset itself was used for debugging purposes, but did not influence the design process. Regardless, only a single dataset was used, which has also been used in similar prior research. It is possible that this dataset lends itself well for such research and does not represent other datasets at all. Since, the main result of this research is the technique itself, rather than the single case study performed, this does not affect the external validity of this study greatly.

Lastly, the reliability of the study should be high. Although the dataset itself is not open, any steps performed in this study have been described and all the tooling and code used to perform

any of these steps, including pre-processing, are open source.

9.4 Future Work

In the previous sections of this chapter and the evaluation done in Section 8.6, various points of improvement have been found. These points of improvement provide starting points for possible future works, continuing on the concepts presented in this study.

The first of such improvements is presented in Section 8.6, where the visual mapping proposed in the design and various points related to the implementation are mentioned. By recreating or iterating on this implementation and adding these proposed improvements could lead to a new research where the effect of said improvements are evaluated. This includes exploring further interaction methods, such as moving through the city. Furthermore, better transformation methods to obtain an Interaction Log may be explored.

Alternatively, the current implementation can be used for further evaluation on alternative datasets, this would allow for finding additional points of improvement of the design and implementation as well as strengthen the external validity of InteractionCity.

During development, drafts were made for a visualization technique similar to InteractionCity, but with support for time. This would allow the user to view the city during a certain time-slice, such as the December holiday season, and view anomalies that would not occur when aggregated on the whole dataset.

A different approach can also be taken, where the concepts in this study; combining Visual Analytics, Software Architecture, and Process Mining and Interaction Logs, are re-applied to an alternative type of visualization, e.g. through an alternative metaphor.

The interaction logs can be further explored, either through further formalization of the log specification, comparison of the Interaction Log to event-based logs, such as XES and OCEL, and possibly explore further applications of interaction logs.

Part V

Additional

Bibliography

[1] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice (Third Edition)*. Addison-Wesley, 2012.

[2] Mark Bruls, Kees Huizing, and Jarke J van Wijk. "Squarified treemaps". In: *Data visualization 2000*. Springer, 2000, pp. 33–42.

[3] Bas Cornelissen et al. "Understanding execution traces using massive sequence and circular bundle views". In: *15th IEEE International Conference on Program Comprehension (ICPC'07)*. IEEE. 2007, pp. 49–58.

[4] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. "Assessing the bus factor of git repositories". In: *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE. 2015, pp. 499–503.

[5] Dusanka Dakic et al. "Business Process Mining Application: A Literature Review." In: *Annals of DAAAM & Proceedings* 29 (2018).

[6] Lakshitha De Silva and Dharini Balasubramaniam. "Controlling software architecture erosion: A survey". In: *Journal of Systems and Software* 85.1 (2012), pp. 132–151.

[7] Stephan Diehl. *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer Science & Business Media, 2007.

[8] Stéphane Ducasse and Damien Pollet. "Software architecture reconstruction: A process-oriented taxonomy". In: *IEEE Transactions on Software Engineering* 35.4 (2009), pp. 573–591.

[9] Stefan Esser and Dirk Fahland. "Multi-dimensional event data in graph databases". In: *Journal on Data Semantics* 10.1 (2021), pp. 109–141.

[10] Dirk Fahland. *Multi... Process Mining, Behavior is not isolated in a case.* URL: `https://multiprocessmining.org/blog/`.

[11] David Garlan. "Software architecture: a roadmap". In: *Proceedings of the Conference on the Future of Software Engineering*. 2000, pp. 91–101.

[12] Cw Christian Günther. "Process mining in flexible environments". In: 2009.

[13] R Hoving. "How to Make the SOK Fit Web Applications?" MA thesis. 2014.

[14] TGC Ipskamp. "A graph-based approach to capture software behavior in architecture". MA thesis. 2018.

[15] Tijmen de Jong and Jan Martijn EM van der Werf. "Process-mining based dynamic software architecture reconstruction". In: *Proceedings of the 13th European Conference on Software Architecture-Volume 2*. 2019, pp. 217–224.

[16] Daniel Keim et al. “Visual analytics: Definition, process, and challenges”. In: *Information visualization*. Springer, 2008, pp. 154–175.

[17] M Klijs. “Interaction Oriented Architecture: A choreography-based design method for component-based software systems”. MA thesis. 2021.

[18] Philippe Kruchten, Henk Obbink, and Judith Stafford. “The past, present, and future for software architecture”. In: *IEEE software* 23.2 (2006), pp. 22–30.

[19] Robert van Langerak, Jan Martijn EM van der Werf, and Sjaak Brinkkemper. “Uncovering the runtime enterprise architecture of a large distributed organisation”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2017, pp. 247–263.

[20] Thomas Panas et al. “Communicating software architecture using a unified single-view visualization”. In: *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*. IEEE. 2007, pp. 217–228.

[21] Process and Data Science Group (PADS). *OCEL Standard*. URL: `http://ocel-standard.org/`.

[22] *ProM: Process Mining framework*. URL: `https://svn.win.tue.nl/trac/prom/`.

[23] CJM Quik. “Visualising Software Dynamics through Architecture Mining”. MA thesis. 2019.

[24] RM Rooimans. “Architecture Mining with ArchitectureCity”. MA thesis. 2017.

[25] Nick Rozanski and Eoin Woods. *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Addison-Wesley, 2012.

[26] Mojtaba Shahin, Peng Liang, and Muhammad Ali Babar. “A systematic review of software architecture visualization techniques”. In: *Journal of Systems and Software* 94 (2014), pp. 161–185.

[27] Mark Sullivan and Ram Chillarege. “Software defects and their impact on system availability-a study of field failures in operating systems”. In: *Digest of Papers. Fault-Tolerant Computing: The Twenty-First International Symposium*. IEEE Computer Society. 1991, pp. 2–3.

[28] Alexandru C Telea. *Data visualization: principles and practice*. CRC Press, 2014.

[29] Wil Van Der Aalst. “Data science in action”. In: *Process mining*. Springer, 2016, pp. 3–23.

[30] Wil Van Der Aalst. “Process mining”. In: *Communications of the ACM* 55.8 (2012), pp. 76–83.

[31] Eric Verbeek and Wil van der Aalst. *IEEE 1849-2016 XES Standard*. URL: `https://xes-standard.org/`.

[32] Jan Martijn EM van der Werf et al. “All that Glitters Is Not Gold”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2021, pp. 141–157.

[33] Richard Wettel and Michele Lanza. “CodeCity: 3D Visualization of Large-Scale Software”. In: *Companion of the 30th International Conference on Software Engineering*. ICSE Companion ’08. Leipzig, Germany: Association for Computing Machinery, 2008, 921–922. ISBN: 9781605580791. DOI: `10.1145/1370175.1370188`. URL: `https://doi.org/10.1145/1370175.1370188`.

[34] Roelf J. Wieringa. *Design science methodology for information systems and software engineering*. Undefined. 10.1007/978-3-662-43839-8. Netherlands: Springer, 2014. ISBN: 978-3-662-43838-1. DOI: `10.1007/978-3-662-43839-8`.

[35] Moe Thandar Wynn et al. “ProcessProfiler3D: A visualisation framework for log-based process performance comparison”. In: *Decision Support Systems* 100 (2017), pp. 93–108.

Appendices

A Running Example Interaction Log

```
<?xml version="1.0" encoding="utf-8" ?>
<log xes.version="1849-2016" xes.features="nested-attributes"
  xmlns="http://www.xes-standard.org/">
  <trace>
    <string key="case" value="1" />
    <event>
      <string key="source:timestamp" value="2022-10-01T10:00:01" />
      <string key="source:location_org" value="com.example" />
      <string key="source:location_project" value="TodoApp" />
      <string key="source:location_namespace" value="Frontend" />
      <string key="source:location_class" value="Selection" />
      <string key="source:action" value="send" />
      <string key="source:message" value="Change" />
      <string key="target:timestamp" value="2022-10-01T10:00:02" />
      <string key="target:location_org" value="com.example" />
      <string key="target:location_project" value="TodoApp" />
      <string key="target:location_namespace" value="Frontend" />
      <string key="target:location_class" value="ListView" />
      <string key="target:action" value="receive" />
      <string key="target:message" value="Change" />
    </event>
    <event>
      <string key="source:timestamp" value="2022-10-01T10:00:03" />
      <string key="source:location_org" value="com.example" />
      <string key="source:location_project" value="TodoApp" />
      <string key="source:location_namespace" value="Frontend" />
      <string key="source:location_class" value="ListView" />
      <string key="source:action" value="send" />
      <string key="source:message" value="Update" />
      <string key="target:timestamp" value="2022-10-01T10:00:04" />
      <string key="target:location_org" value="com.example" />
      <string key="target:location_project" value="TodoApp" />
      <string key="target:location_namespace" value="Frontend" />
      <string key="target:location_class" value="ListManager" />
      <string key="target:action" value="receive" />
      <string key="target:message" value="Update" />
    </event>
    <event>
      <string key="source:timestamp" value="2022-10-01T10:00:05" />
      <string key="source:location_org" value="com.example" />
      <string key="source:location_project" value="TodoApp" />
      <string key="source:location_namespace" value="Frontend" />
      <string key="source:location_class" value="ListManager" />
```

```
      <string key="source:action" value="send" />
      <string key="source:message" value="Update" />
      <string key="target:timestamp" value="2022-10-01T10:00:06" />
      <string key="target:location_org" value="com.example" />
      <string key="target:location_project" value="TodoApp" />
      <string key="target:location_namespace" value="Backend" />
      <string key="target:location_class" value="ListManager" />
      <string key="target:action" value="receive" />
      <string key="target:message" value="Update" />
    </event>
    <event>
      <string key="source:timestamp" value="2022-10-01T10:00:07" />
      <string key="source:location_org" value="com.example" />
      <string key="source:location_project" value="TodoApp" />
      <string key="source:location_namespace" value="Frontend" />
      <string key="source:location_class" value="ListManager" />
      <string key="source:action" value="send" />
      <string key="source:message" value="IsLoading" />
      <string key="target:timestamp" value="2022-10-01T10:00:08" />
      <string key="target:location_org" value="com.example" />
      <string key="target:location_project" value="TodoApp" />
      <string key="target:location_namespace" value="Frontend" />
      <string key="target:location_class" value="ListView" />
      <string key="target:action" value="receive" />
      <string key="target:message" value="IsLoading" />
    </event>
    <event>
      <string key="source:timestamp" value="2022-10-01T10:00:09" />
      <string key="source:location_org" value="com.example" />
      <string key="source:location_project" value="TodoApp" />
      <string key="source:location_namespace" value="Backend" />
      <string key="source:location_class" value="ListManager" />
      <string key="source:action" value="send" />
      <string key="source:message" value="IsAuthenticated" />
      <string key="target:timestamp" value="2022-10-01T10:00:10" />
      <string key="target:location_org" value="com.example" />
      <string key="target:location_project" value="TodoApp" />
      <string key="target:location_namespace" value="Backend" />
      <string key="target:location_class" value="LoginManager" />
      <string key="target:action" value="receive" />
      <string key="target:message" value="IsAuthenticated" />
    </event>
    <event>
      <string key="source:timestamp" value="2022-10-01T10:00:11" />
      <string key="source:location_org" value="com.example" />
      <string key="source:location_project" value="TodoApp" />
      <string key="source:location_namespace" value="Backend" />
```

```
      <string key="source:location_class" value="LoginManager" />
      <string key="source:action" value="send" />
      <string key="source:message" value="UpdateState" />
      <string key="target:timestamp" value="2022-10-01T10:00:12" />
      <string key="target:location_org" value="com.example" />
      <string key="target:location_project" value="TodoApp" />
      <string key="target:location_namespace" value="Backend" />
      <string key="target:location_class" value="LoginManager" />
      <string key="target:action" value="receive" />
      <string key="target:message" value="UpdateState" />
    </event>
    <event>
      <string key="source:timestamp" value="2022-10-01T10:00:13" />
      <string key="source:location_org" value="com.example" />
      <string key="source:location_project" value="TodoApp" />
      <string key="source:location_namespace" value="Backend" />
      <string key="source:location_class" value="LoginManager" />
      <string key="source:action" value="send" />
      <string key="source:message" value="IsAuthenticated" />
      <string key="target:timestamp" value="2022-10-01T10:00:14" />
      <string key="target:location_org" value="com.example" />
      <string key="target:location_project" value="TodoApp" />
      <string key="target:location_namespace" value="Backend" />
      <string key="target:location_class" value="ListManager" />
      <string key="target:action" value="receive" />
      <string key="target:message" value="IsAuthenticated" />
    </event>
    <event>
      <string key="source:timestamp" value="2022-10-01T10:00:15" />
      <string key="source:location_org" value="com.example" />
      <string key="source:location_project" value="TodoApp" />
      <string key="source:location_namespace" value="Backend" />
      <string key="source:location_class" value="ListManager" />
      <string key="source:action" value="send" />
      <string key="source:message" value="Update" />
      <string key="target:timestamp" value="2022-10-01T10:00:16" />
      <string key="target:location_org" value="com.example" />
      <string key="target:location_project" value="TodoApp" />
      <string key="target:location_namespace" value="Outsource" />
      <string key="target:location_class" value="Database" />
      <string key="target:action" value="receive" />
      <string key="target:message" value="Update" />
    </event>
    <event>
      <string key="source:timestamp" value="2022-10-01T10:00:17" />
      <string key="source:location_org" value="com.example" />
      <string key="source:location_project" value="TodoApp" />
```

```
      <string key="source:location_namespace" value="Outsource" />
      <string key="source:location_class" value="Database" />
      <string key="source:action" value="send" />
      <string key="source:message" value="IsUpdated" />
      <string key="target:timestamp" value="2022-10-01T10:00:18" />
      <string key="target:location_org" value="com.example" />
      <string key="target:location_project" value="TodoApp" />
      <string key="target:location_namespace" value="Backend" />
      <string key="target:location_class" value="ListManager" />
      <string key="target:action" value="receive" />
      <string key="target:message" value="IsUpdated" />
    </event>
    <event>
      <string key="source:timestamp" value="2022-10-01T10:00:19" />
      <string key="source:location_org" value="com.example" />
      <string key="source:location_project" value="TodoApp" />
      <string key="source:location_namespace" value="Backend" />
      <string key="source:location_class" value="ListManager" />
      <string key="source:action" value="send" />
      <string key="source:message" value="IsUpdated" />
      <string key="target:timestamp" value="2022-10-01T10:00:20" />
      <string key="target:location_org" value="com.example" />
      <string key="target:location_project" value="TodoApp" />
      <string key="target:location_namespace" value="Frontend" />
      <string key="target:location_class" value="ListView" />
      <string key="target:action" value="receive" />
      <string key="target:message" value="IsUpdated" />
    </event>
  </trace>
</log>
```