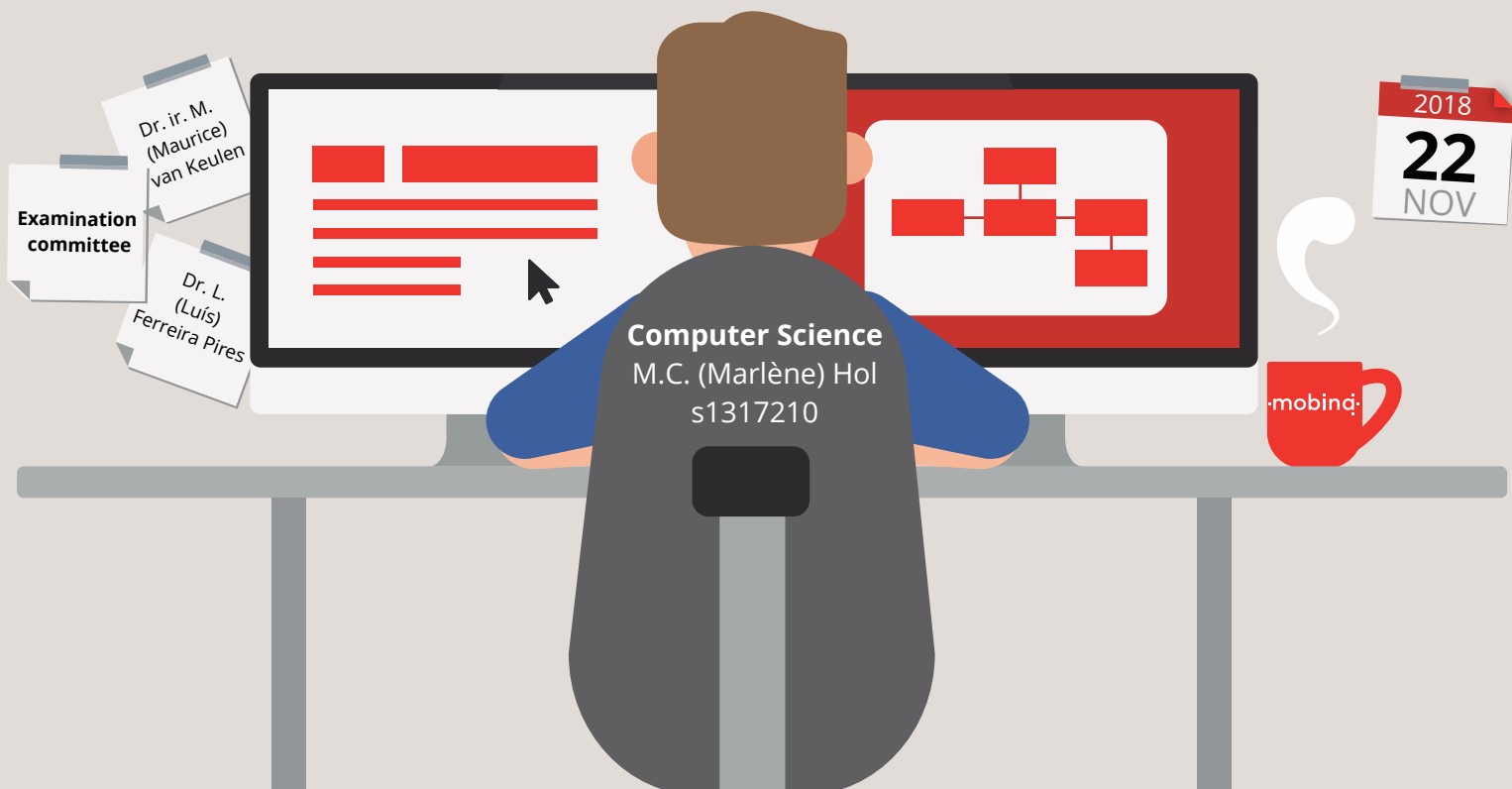


UNIVERSITY
OF TWENTE.



USING **USER WORKFLOW ANALYSIS** TO CREATE INSIGHTS IN CONTENT-INTENSIVE APPLICATIONS



Combining **Process Mining** and **Model-Driven Engineering**
to create a reusable, scalable and user-friendly solution

Cover illustration created with a vector by Freepik

Preface

This thesis completes my exciting time at the University of Twente. It has been an amazing journey from starting here as a freshman in 2012, to becoming an alumnus now. For me, the University of Twente will always be a special place. It is the place where I learnt so many interesting things, developed so many different skills, where I founded my company Mobina, where I found love, made great friends and so much more. But all good things come to an end. However, this is not a sad ending. I'm looking forward to putting all my time and energy in Mobina, and hopefully to make it the company we all believe it can be.

I would like to take this opportunity to thank several people. First, I would like to thank my supervisors Maurice van Keulen and Luís Ferreira Pires for their valuable feedback on my thesis. I would like to thank Maurice for always giving me the opportunity to explore the topics I was really interested in and making it possible to combine this with Mobina. I would like to thank Luís for always being critical, not only at my design and implementation, but also by giving me valuable textual feedback.

Next, I would like to thank everybody at Mobina, with a special thanks to some of them. First, I would like to thank Jasper Boot, René Hol, Valerija Olsevska, Jochem Verburg, and Hans Wortmann for participating in the data collection. I would also like to thank René, Valerija, and Jochem for using their expertise in the Mobina content in the validation. Additionally, I want to thank Jasper for his technical support when we developed the logging framework necessary to perform this project. Because of his background in Business Information Technology, Jochem also proved himself valuable as company supervisor. I would like to thank him for all his feedback on my project, both content-wise and textual. Finally, I would like to thank René and Jochem for giving me all the freedom to perform this project and giving me the opportunity to steer this project in a direction that I deemed not only useful for Mobina, but that is also aligned with my personal interest and study background. Their unconditional trust in the project and the road I was taking greatly motivated me.

Finally, I would like to thank my friends and family for their (moral) support during my thesis. I could not have done it without them. I would like to thank Meike Nauta and Caspar Schutijser for taking the time to review my report. I would like to thank Mart Oude Weernink for his creativity that led to the cover of my thesis. A very special thanks to my parents, not only for their support during my thesis, but also for the support they have given me my entire life. I would like to especially thank my father for always giving me career advice, leading to me graduating in Computer Science and us building an amazing company together with the rest of the Mobina team. I would like to thank my mother for always having my best interests at heart, making it possible for me to achieve everything I am proud of. Finally, I would like to thank Jochem for his support over the past few months, for always encouraging me to be the best version of myself, and for always looking out for me, especially when I was not able to do so myself.

Abstract

Traditional methods for evaluating software are often less suitable for content-intensive applications because they are not always able to track the steps of a user across the entire application, because they are less suitable for continuous improvement due to their lack of scalability, and because the observed behavior might be different from reality. We introduce user workflow analysis as a solution to help content and software developers (user workflow analyzers) of content-intensive applications (referred to as System Under Analysis or SUA) to gain additional insights in the usage of the application that can't be acquired through traditional methods.

We present a design and prototype implementation of the User Workflow Analysis Tool (UWAT). This tool successfully implements user workflow analysis by combining the strength of Process Mining (PM) in extracting patterns from event log data and the strength of Model-Driven Engineering (MDE) in standardized and automatic transformations. This research shows that MDE, due to its standardization power, can mitigate the shortcoming of PM that it is often left up to user to configure the plugins and interpret the results. In this design, only the implementer of the UWAT needs to understand the PM configuration and results, and all other user workflow analyzers can benefit from this in a standardized, automated and user-friendly way. No references were found that this combination was used for this purpose before.

The UWAT itself consists of five metamodels: SUA metamodel, plugin specification metamodel, user specification metamodel, Process Mining metamodel, and result metamodel. All metamodels are set-up as generic as possible, so they can easily be used for multiple SUAs. Four transformations are introduced: to create the SUA model instance, to translate the input of the user workflow analyzer to a user specification model instance and the instructions for PM, to process the result of PM to useful results for the user workflow analyzers based on the user specification, and finally to translate the result model instance to the output data that can be used for visualization of the results. The PM execution happens externally, but the UWAT handles the call to these external plugins. In this way, users of the UWAT can benefit from external knowledge in an area that is rapidly evolving. The visualization for the user workflow analyzers is separated from the UWAT, so the SUA implementer has a lot of flexibility in how to present the results.

For the prototype implementation and validation, a case study is done for a content-intensive application, Mobina. The Mobina team created their own modelling technique for the complex content. They are expected to benefit from user workflow analysis to validate the modelling technique and the interface supporting this. The design was validated using two types of expert opinion: user validation and technical review. Potential user workflow analyzers were interviewed to identify the added value of user workflow analysis and to assess the requirements relevant for user workflow analyzers. The technical review is a critical review of this design, and where necessary its implementation, to validate to what extent the requirements of the architecture were fulfilled.

This thesis shows that using user workflow analysis can provide a lot of added value to SUAs. By using the UWAT, the user workflow analyzers get more insights in the behavior of the users in the SUA and the effects of their work on their users. Eventually, this will lead to a better user experience and satisfied customers. Simple examples already have added value, but this could be even more with analyses that focus on the structure of the content and the software.

This research successfully showed how by combining PM and MDE, different user workflow analyzers can benefit from analyses in a reusable, scalable and user-friendly way. This combination of PM and MDE led to a user-friendly solution where the user workflow analyzers are not bothered with the implementation details. New functionality and analyses can easily be added which makes the solution flexible and scalable. Due to the generic setup of the metamodels, most of the metamodels can be reused for different SUAs. Some parts of the transformations need to be implemented for each SUA separately, so the SUA implementer has a lot of flexibility and can implement all desired analyses. As soon as the necessary PM support is there, this solution can also be fully automated.

User workflow analysis as presented shows a lot of potential, but we expect this can be even more. It will be interesting to see whether other techniques, e.g., machine learning, can improve or extend the user workflow analyses. Other types of applications than content-intensive applications could also benefit from user workflow analysis and this design, so it is useful to research this further. Finally, it will be very interesting to bring this to the next step of the design cycle. The design presented here contains all the core aspects, but there are also several remaining opportunities. There are possibilities to standardization and share one UWAT implementation amongst SUAs. When researching these possibilities more in-depth, an even better support for user workflow analysis can be established.

Table of Contents

Preface	3
Abstract.....	5
Table of Contents.....	7
Glossary.....	10
1 Introduction.....	11
1.1 Problem statement.....	11
1.2 Research questions	12
1.3 Research design.....	13
1.3.1 Scope	13
1.3.2 Approach	14
1.4 Report structure	14
2 Background and related work.....	15
2.1 Model-Driven Engineering (MDE)	15
2.1.1 Concepts	15
2.1.2 Use cases.....	17
2.1.3 MDE in practice	17
2.2 Process Mining (PM).....	18
2.2.1 Techniques	18
2.2.2 PM in software engineering.....	18
2.3 Combining PM and MDE.....	19
3 Case study: Mobina	20
3.1 Mobina application.....	20
3.2 Relevance.....	20
3.3 Mobina components.....	21
4 Solution.....	24
4.1 Requirements	24
4.1.1 Functional requirements	24
4.1.2 Non-functional requirements	24
4.1.3 Domain requirements.....	25
4.2 High-level architecture	25
4.3 Techniques	26
5 (Meta)models and transformations.....	27
5.1 Overview.....	27
5.2 Metamodels	27
5.2.1 SUA metamodel.....	27
5.2.2 Plugin specification metamodel.....	29
5.2.3 User specification metamodel.....	30
5.2.4 Process Mining metamodel.....	31

5.2.5	Result metamodel	33
5.3	Transformations	35
5.3.1	Data-to-SUA transformation	35
5.3.2	Preparation-to-specification transformation	35
5.3.3	Process-to-result transformation	36
5.3.4	Result-to-dashboard transformation	36
6	Implementation prototype	38
6.1	Scope	38
6.2	Tools	38
6.3	Preparation	39
6.3.1	Preparing the event log	39
6.3.2	Data-to-SUA	40
6.4	User preparation	41
6.5	UWAT and PM implementation	42
6.5.1	Preparation-to-specification	43
6.5.2	Process input	45
6.5.3	Process output	45
6.5.4	Process-to-result	47
6.5.5	Result-to-dashboard	48
6.6	Result dashboard	49
7	SUA guidelines	51
7.1	Business objectives	51
7.1.1	Determining the business objectives	51
7.1.2	Case study	51
7.2	Logging of the software	52
7.2.1	Logging data	52
7.2.2	Logging framework	52
7.2.3	Case study	53
7.3	Data generation	55
7.3.1	Experiment scoping	56
7.3.2	Experiment planning	56
7.3.3	Experiment operation	57
7.3.4	Presentation & package	58
7.3.5	Case study	58
8	Validation	60
8.1	Approach	60
8.1.1	User validation	60
8.1.2	Technical review	61
8.2	Validation results	61

8.2.1	Functional requirements	61
8.2.2	Non-functional requirements	63
8.2.3	Domain requirements.....	66
8.2.4	Usefulness of user workflow analysis.....	66
8.3	Summary	66
9	Discussion	68
9.1	Added value user workflow analysis	68
9.2	Setup of the UWAT	68
9.2.1	Combination of PM and MDE.....	68
9.2.2	Reusability	69
9.2.3	Separation visualization and UWAT.....	70
9.2.4	User friendliness	71
9.2.5	Scalability	71
9.3	Implications of proof of concept.....	72
9.4	Validity.....	72
10	Conclusion	74
10.1	Summary	74
10.2	Implications and recommendations for content-intensive applications	75
10.3	Scientific contributions and future research.....	76
	References	78
	Appendix A. Business objectives Mobina	80
	Appendix B. Data collection document for the participants.....	82

Glossary

Data-to-SUA transformation	Transformation that translates the SUA data to an instance of the SUA metamodel.
Mobina	The content-intensive application user for the case study.
Model-Driven Engineering (MDE)	Methodology that is used to implement the UWAT.
Plugin specification (meta)model	(Meta)model that contains the specification of a PM plugin.
Preparation-to-specification transformation	Transformation to translate the specification of the user in a preparation interface to an instance of the user specification metamodel.
Process Mining (PM)	Technique that is used to extract the patterns from the event log data of the SUA.
Process Mining (meta)model	(Meta)model that contains the outcomes of the PM plugins of the user specification.
Process-to-result transformation	Transformation that translates a PM model, a user specification model, and a SUA model to an instance of the result metamodel.
Reference model	Content element of the case study Mobina.
Result (meta)model	(Meta)models that contains the results of the analyses.
Result-to-dashboard transformation	Transformation that translates a result model to a textual representation that can be used for visualization.
SUA data	Data about the SUA that is relevant for the user workflow analysis.
SUA (meta)model	(Meta)model that contains all the specifics of the SUA
SUA owner	The person responsible for implementing the UWAT in the SUA development process.
System Under Analysis (SUA)	The content-intensive application that is being analyzed.
User specification (meta)model	(Meta)model that contains the specification of the user workflow analyzer on what to analyze.
User workflow	The sequence of interactions the user has with the entire artifact.
User workflow analyzer	The user of the UWAT. This includes the content and software developers of the SUA.
User Workflow Analysis Tool (UWAT)	The artifact of this research. The UWAT is a tool that can be used by user workflow analyzers to analyze the user workflow of the SUA.

1 Introduction

This chapter introduces this thesis. Firstly, the problem statement is introduced with a strong focus on content-intensive applications. This is followed up by the introduction of the research questions. Then, the research design is presented including the scope of this thesis. Finally, the rest of the report is introduced.

1.1 Problem statement

Anyone who develops a software product has the goal to develop a product that optimally supports the user. If the user has the feeling that he/she can use the software effectively, this will lead to higher user satisfaction. It is therefore widely accepted that the user should play an important role in software development processes to make sure the user can get the most out of the developed software [1]–[3].

This is especially the case for content-intensive applications, which are applications containing (complex) content and where the software is designed and developed to support this content. In this case, the structure of the software and its content, i.e., the way the software and its content are designed and modelled, determines a large part of the user satisfaction. The content needs to be modelled and structured so that the user can always understand the software and find its way through it.

Currently, off-the-shelf software is often developed based on the experience and expectations of the company who develops the software. The user often does not (systematically) participate in the software development process, and when the user is involved, often more traditional methods focusing on the graphical user interface (e.g., whether the user can easily save the progress on the screen) or using statistical facts about the software (e.g., the number of errors) are used to test software usability [4], [5]. Involving the user through these traditional methods already improves the quality of software significantly and will improve system usage [6], [7]. However, these methods are often less usable nor enough for content-intensive applications for three reasons:

1. These methods are often more focused on details and the graphical user interface. This makes them unsuitable for determining whether the software and its content are structured correctly and, more importantly, it makes them unsuitable to discover the different steps or tasks the user is taking across the entire application. Especially in content-intensive applications, the actions that are not directly related to the user interface and are not necessarily in the same view are crucial to determine the next steps and improvements in the software and its content.
2. These traditional methods are often used for the initial design, and not for continuously improving the software. Involving the users in the entire development process is often not scalable, especially in the case of user testing where the actual person and its actions must be tracked. Every time the user is involved leads to additional expenses, which adds a lot of extra costs to the software development process [8], [9]. Consequently, this also often means that only a small part of the (potential) users are involved instead of the entire target population. Due to this poor scalability, these methods are less suitable for continuously improving applications.
3. In the case of content-intensive applications, learning from the actual behavior is crucial, since the users then really start to use the (often complex) content. Some time is necessary to grasp the content, and this cannot easily be staged in a short time period to keep user testing affordable. Also, the setting of user testing is often artificial, which can influence the actual interactions with the content. Therefore, a solution where one can learn from actual behavior is crucial when looking into content-intensive applications.

In the past few years, a new approach for evaluating software has been developed, namely user workflow analysis. In user workflow analysis the log data is evaluated which includes all the different interactions the user has with the entire artifact, in this case the software application. These interactions can be to achieve a specific goal, e.g., to reach a certain state or perform a specific action, or simply a sequence of steps with no immediate goal. User workflow analysis provides the opportunity to capture the actual process automatically and to see how this process relates to the expected or desired process. Emerging techniques like data mining, machine learning, and process mining enable and improve this approach to analysis.

User workflow analysis can therefore be very useful for content-intensive applications, especially since it seems to fill the gaps left by more traditional methods. Firstly, it does not focus on the details but on how the user interacts with the application. Secondly, it is capable of capturing behavior across the entire application and not just a small part of the application. Therefore, it is also suitable for analyzing not only functionality, but also the content that could highly influence each interaction. Additionally, since it tracks actual behavior the results are not influenced and there is no artificial setting. Finally, it has the potential to become fully automated which would make it scalable. Because of these reasons, user workflow analysis is expected to help identify where the predicted workflow differs from the discovered workflow and whether the content and the software are properly structured.

Several techniques are available that can (partially) implement user workflow analysis. Based on the goals of the analysis, and the interest and experience of the developer, everyone should decide for themselves which techniques to use. In this research, a combination of two techniques is chosen to implement user workflow analysis: Process Mining (PM) and Model-Driven Engineering (MDE). These techniques have the potential to create a solution which can be applied to all content-intensive applications in a generic way. Additionally, we expect these techniques can be fully automated, which makes them scalable and suitable for continuously improving software. Finally, we expect that these techniques help present the results in ways that anyone can analyze the results, not only the software developers. Due to these reasons and previous experience of the researchers, these techniques are selected for this research.

1.2 Research questions

In this research, we investigate whether user workflow analysis can indeed give additional insights in content-intensive applications that are not possible with traditional methods. This leads to our first main research question:

1. *How can user workflow analysis help content and software developers of content-intensive applications gain additional insights in the content and the software?*

As mentioned before, user workflow analysis can be implemented in many ways. In this research, we inspect the implementation with PM and MDE. The combination of these two techniques has not been extensively researched yet. This leads to our second main research question:

2. *How can Process Mining and Model-Driven Engineering be used in combination to implement user workflow analysis?*

The expected potential of these techniques described above lead to three sub questions of the second research question:

- To what degree can this combination be used to implement user workflow analysis in a generic way to make it usable for all content-intensive applications?
- To what degree can this combination be used to implement user workflow analysis in a scalable way?
- To what degree can this combination be used to implement user workflow analysis in a way that content and software developers can use the results without needing to know the details of the implementation?

1.3 Research design

In this research, an architecture to perform user workflow analysis with PM and MDE has been designed. A prototype of the tool has been implemented and validated with a case study.

1.3.1 Scope

Figure 1 shows how the tool designed in this research is integrated in the development of content-intensive applications. The research artifact is the *User Workflow Analysis Tool* (UWAT). Content and software developers of content-intensive applications will use this tool to analyze the user workflow of content-intensive applications: *user workflow analyzers*. The content-intensive application is the *System Under Analysis* (SUA). The SUA is not part of the UWAT, merely the object that is being analyzed. The *SUA owner* is responsible for implementing the architecture in the SUA development process.

The SUA user interacts with the SUA. These interactions are logged in the *log data* and are the input for the UWAT, together with other relevant information of the SUA, namely the *SUA data*. Based on these data, the UWAT can generate relevant results for the user workflow analyzer, which can then use these results to decide whether to improve the SUA.

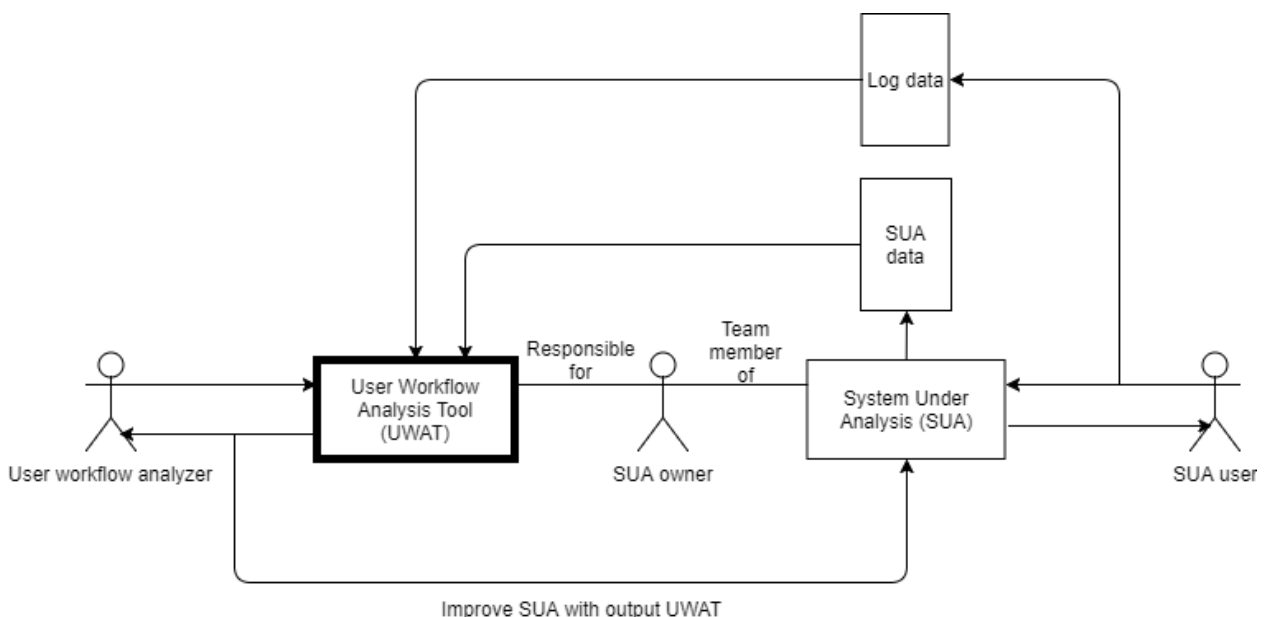


Figure 1 Role of the UWAT

This research focuses on designing and analyzing an UWAT and did not aim to create useful analyses for the SUA. This research discusses which collected data is necessary for user workflow analysis, and this is applied to a case study. However, the architecture designed and implemented assumes that these data are available.

1.3.2 Approach

The research consists of the following three phases:

1. *Design*. In this phase, the requirements of the architecture have been defined, which have been used to design the architecture. The design includes the definition of the (meta)models and transformations as well as the integration of PM and MDE.
2. *Implementation*. In this phase, the architecture has been implemented. This implementation has been used for the validation. The implementation is based on a case study using a content-intensive application.
3. *Validation*. In this phase, the architecture has been validated, and the research questions have been answered. The user workflow analyzers of the case study were involved in the validation.

As mentioned before, the architecture needs log data and actual information about the SUA. Furthermore, the user workflow analyzers should know beforehand what they want to analyze, since this defines what the results should be. Therefore, before implementing the UWAT the business objectives should be clear and data collection should be aligned to support these business objectives. Although they are not important for the UWAT architecture, these are important aspects for the case study and are therefore described in this thesis.

1.4 Report structure

The thesis is further structured as follows. Chapter 2 presents background on the selected techniques and the related work. Chapter 3 presents the case study that is used as an example and for validation. The SUA in this case study is the content-intensive application Mobina.

Chapter 4 gives the requirements and the UWAT high-level architecture. The UWAT is mainly implemented using MDE. Chapter 5 presents the metamodels and transformations of the MDE implementation. Chapter 6 describes the implementation of the prototype. The prototype is implemented for the case study and was used for the validation of the design. Chapter 7 discusses the preparations each SUA owner should do. The implementation for the case study is also discussed.

Chapter 8 describes the validation set-up and results. Chapter 9 discusses the results. Chapter 10 summarizes the results and answers the research questions. It also presents implications and recommendations for the content-intensive applications considering using user workflow analysis and the UWAT. Finally, it presents the scientific contributions and interesting topics for future research.

2 Background and related work

This chapter presents background information on MDE and PM. This chapter also investigates and discusses the application of MDE in practice and PM for extracting the user workflow. Additionally, existing research about the combination of both techniques is discussed.

2.1 Model-Driven Engineering (MDE)

This section explains what MDE is and the concepts behind it. It also describes the cases in which MDE is useful. Last, it presents the strengths and risks of using MDE in practice.

2.1.1 Concepts

The foundation of the Model-Driven Engineering (MDE) methodology is the Model-Driven Architecture (MDA) initiative of the OMG group¹. The motivation of this initiative has been to separate the specification of system functionality from the implementation of that functionality on a specific technology platform [10], [11]. This led to the distinction of Platform Independent Models (PIMs) and Platform Specific Models (PSM). MDA presents three levels of modeling abstraction: PSM, PIM, and Computation Independent Models (CIM) [12], [13]. Figure 2 depicts these levels.

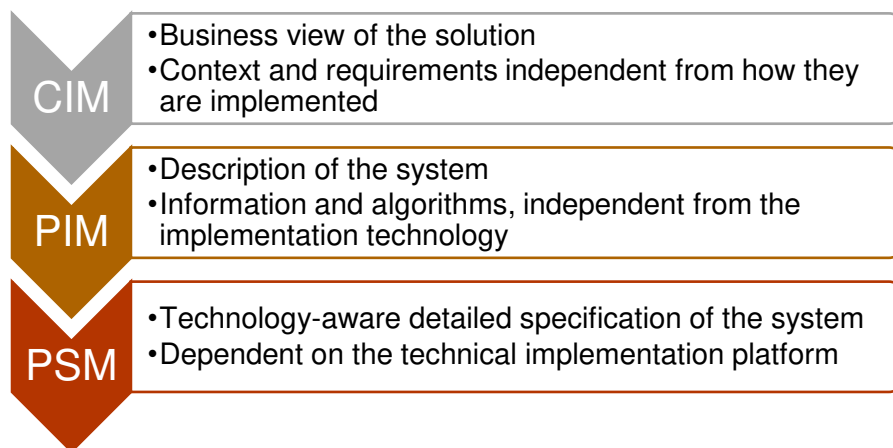


Figure 2 Three levels of modelling abstraction

There are several reasons to separate these levels. Firstly, by abstracting from a specific platform, the correctness of the model can be validated more easily. Secondly, it is easier to produce implementations on different platforms, while having the same structure and behavior. This also means technology can be changed without having to redevelop and redesign the entire application. Finally, it improves the integration and interoperability across systems because they can be defined in platform-independent terms [10].

MDA is just a framework, so it does not incorporate a specific development process or methodology. The broader term Model-Driven Engineering (MDE) embodies a lot of the MDA technologies but combines this with a software development process. MDE employs the general concepts of modelling, independent of a particular standard.

Models are the foundation of MDE. In MDE a metamodel defines the abstract syntax of a language that allows models to be defined. This could lead to an unlimited level of models. Figure 3 shows the four layers of metamodeling that are often used with MDE [12].

¹ <http://www.omg.org/mda/>

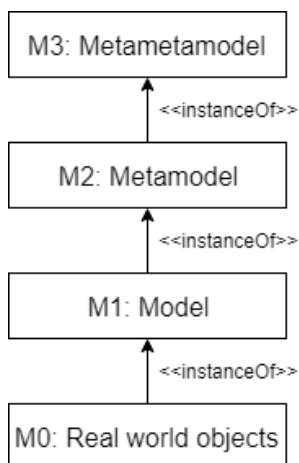


Figure 3 Four layers of metamodeling often used in MDE

Model transformations are an important technique in MDE. The idea is that one can automatically generate other models or text based on an input model and the metamodel definitions. A developer can manually write these transformations, but they can also be produced automatically using some higher-level mapping rules between models. When using the latter approach, the transformations are actually models.

Figure 4 shows how transformations work in MDE, and how this relates to models and metamodels. A source model is translated to a target model, which can be, for example, a PIM that is transformed to a PSM. The transformation uses as input the source model and the transformation definition. The transformation definition is based on the metamodel of the source and the target model. This transformation therefore knows which elements are in both models. The transformation definition describes how an element in the source model should be transformed to an element of the target model. This transformation definition, the source metamodel, and the target metamodel are all instances of a metametamodel.

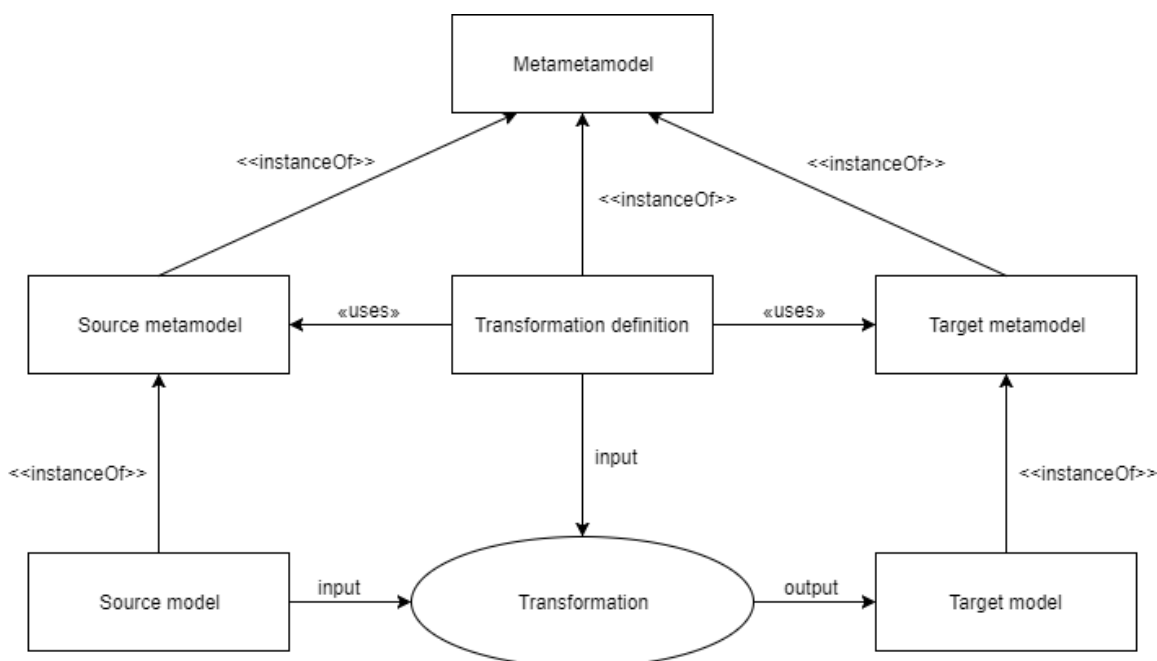


Figure 4 Transformation in MDE

Metamodeling allows the definition of Domain-Specific Languages (DSLs). The purpose of DSLs is to offer a language that domain experts understand. This allows domain experts to participate in the modelling and software development process, since the DSL abstracts from the technical details.

2.1.2 Use cases

Since OMG launched MDA in 2001, many usages of MDA and MDE have been presented both in research and in practice. Three main application scenarios can be identified: (1) automating software development, (2) system interoperability, and (3) reverse engineering [12].

In the first scenario, software code can be generated automatically from the models. Because metamodels are available, the (software) developer exactly knows what information can be in the models. Model-to-model transformations and model-to-text transformations can be created based on these possible model elements. To create a new piece of code, one creates a new instance of the metamodel (a model) that contains the specific information, and code can automatically be generated for this model. This technique is especially useful in case code needs to be generated often. For one-off tasks the overhead of modelling and writing the transformations is too much, but for repetitive tasks this can be very beneficial. Another advantage is that everybody can create such a model, and software developers are in theory not necessary anymore to create the whole software, only for writing and updating the transformations.

In the second scenario, there are different systems with a similar purpose which each have their own metamodel. In this scenario, also a pivot metamodel is created which is a metamodel that covers all the different systems. Then, transformations can be written and executed from the metamodel of the different systems to the pivot metamodel and vice versa. In this way, a model of each system can be translated to a model of another system via the pivot metamodel. The alternative is to write a transformation between all metamodels of the different systems directly. However, using n systems, one needs to write $n*(n-1)$ transformations. When a new system is introduced, again n transformations need to be written. This quickly explodes. By using a pivot metamodel, only two transformations need to be added for a new system (from and to the pivot metamodel) and the new system can directly be transformed to any of the other systems.

In the third scenario, MDE is used to extract the models of already existing systems, often legacy systems. One starts with some low-level models and based on these models one can extract higher level models. From the discovered higher-level models, a new system can be generated using the first scenario. In this way, a legacy system can be modernized.

2.1.3 MDE in practice

There are many different reasons to apply MDE in a project or organization. In different studies about MDE in practice, the most named benefits are: a clear (well-documented) software architecture, conceptual simplicity, efficient implementation, high scalability, flexibility, higher productivity, and improved communication within development teams and with external stakeholders [14]–[16]. Especially for labor-intensive and error-prone development tasks, MDE is deemed useful [16], [17].

However, research has also shown limitations when using MDE in practice. A limitation that is very prominent in other research is the scaling-up of MDE. Most success stories had a small DSL, i.e., it is implemented in a narrow domain [14], [18]. The tools and techniques are often also not suitable for usage across platforms. Most tools are bound to specific platforms [16]. Besides this, version control and change management in MDE seems to be under-developed [17], [18]. There is often a tradeoff between flexibility and automation. For each situation, the potential benefits should outweigh the potential risks, and the project and organization must fit the current MDE practices.

2.2 Process Mining (PM)

PM is a research discipline that aims to discover, monitor and improve actual processes by extracting knowledge from event logs readily available in today's information systems. The IEEE Task Force on PM has written a manifesto, which presents guiding principles and challenges for PM [19]. A lot of the research about PM and the principles outlined in this research are based on this manifesto. An overview of PM based on the manifesto is also has been published in [20].

2.2.1 Techniques

PM is a technique that is currently rapidly developing. More applications and extensions are discovered on a regular basis. Three main techniques form the foundation of PM: process discovery, conformance checking and enhancement [21], [22].

Process discovery uses an event log as input and automatically constructs a process model from this event log without any a priori information. The resulting process model can be presented in different formats and depends on the algorithm used to discover the process model. The most basic process discovery algorithm is the alpha algorithm, which produces a Petri Net. This is a simple algorithm, which has several limitations like its inability to discover concurrent processes. There are also other (more advanced) algorithms like the region-based approaches, which can discover more complex model structures, and the heuristics miner, which focuses on noise and incompleteness.

Conformance checking uses as input both a process model and an event log. Both are then compared to see if the reality matches the expected or desired behavior. The input model can be a handmade model. Typically, four quality dimensions for comparing models and logs are considered:

- *Fitness*. The more logs can be replayed on the model, the higher the fitness.
- *Simplicity*. The simplicity of the model. A principle called Occam's Razor is often important here. The simplest model that can explain the behavior seen in the log is considered as the best model.
- *Precision*. Whether the model does not allow for too much behavior. If the model is not precise, the model is underfitting; the model over-generalizes the example behavior of the log.
- *Generalization*. If the model does not restrict the behavior too much to the examples of the log. If the model is not general, the model is overfitting; the model is too specific for the example data in the event log.

Conformance checking results can be viewed from two angles: the model does not capture the real behavior (the model is wrong) or reality deviates from the desired model (the actual behavior is not as it should be). In the former viewpoint the model is a descriptive model, in the latter a normative model.

Enhancement can extend or improve the existing process model using the event log. This can be done in several ways. Non-fitting process models can be corrected using diagnostics provided by alignment. Event logs can also contain additional information about, for example, resources, timestamps, and case data. This extra information can be used, for example, in the discovery of roles, construction of social networks, analyzing resource performance, and creating decision trees.

2.2.2 PM in software engineering

Keith et al. [23] observed that PM is an area with great potential for application in the field of software engineering. Thanks to its capacity to discover actual processes, assess their conformity with respect to the official model, and find improvement opportunities, PM has high potential for many applications.

The research on PM in software engineering can roughly be separated in two categories: PM used for the software development process and PM used in the software process, i.e., the process within the product. Since the research in this project focuses on evaluating the software and its content, the latter category is of interest for us.

Van der Aalst et al. [24] present a technique to compare the actual behavior in an information system with the intended or expected behavior. In this approach, delta testing is used to compare the discovered process model with the predefined model, and conformance testing is used to quantify the fit between the actual and predefined model. Poncin et al. [25] present FRASR (Framework for Analyzing Software Repositories), a framework that enables the use of PM techniques on combined data of multiple repositories, like code repositories, the bug tracking system, mail, etc. Using the combined view, more valuable information can be extracted. Van der Aalst et al. [26] present a technique to analyze software in vivo, in other words in their natural habitat. By observing running systems, and collecting and analyzing data of these systems, descriptive models can be generated, and these can be used to respond to failures. Rubin et al. [27] present the results of applying PM techniques on several systems used in the touristic domain. In this case, user interaction is recorded in event logs. Based on these logs, process and user interface flow models are automatically derived.

We also investigated which PM techniques are most useful for evaluating software products. Ailenei et al. [28] developed a set of use cases and validated these use cases by means of expert interviews and a survey. In this thesis, the role of PM can be compared with the role of the process analyst from [28]. The paper shows that especially the use cases related to process discovery and the time perspective are relevant for this kind of research. Organizational aspects and compliance are deemed less useful.

Most of the PM research ends with the results of the techniques and does not mention what happens further with these results. Work has been presented to visualize the results in charts or through simulation [29], [30]. However, one still needs to interpret the results himself.

2.3 Combining PM and MDE

In the literature research so far, PM and MDE are described extensively. However, the combination of both techniques has not been extensively discussed. We found no references that uses MDE to analyze or predefine the PM results. Even though this exact topic has not been studied yet, there is some research that is worth mentioning.

Some research is about using the MDE models for conformance checking. Simonin et al. [31] presents a research where models are automatically designed by using MDE. These models are then used by certain people, e.g., the supervisor, to detect anomalies in the processes discovered by using PM techniques. Bernardi et al. [32] also uses the models created by using MDE as a normative model for the results of PM, in this case to improve the security of web information systems. The normative model is generated based on the specification of the system.

Mazak et al. present a combination of PM and MDE to tackle the MDE limitation that it is now mainly used for prescriptive models. However, to make them beneficial, the models should be updated at runtime. Execution-based model profiling is proposed to tackle this problem [33], [34]. In this case, the runtime character of PM is combined with the (prescriptive) modelling techniques of MDE, to assess whether practice and expectations are really aligned.

3 Case study: Mobina

This chapter introduces the case study of this thesis, the content-intensive application Mobina. First, Mobina is introduced. Then, we motivate why Mobina is a relevant case study as well as how Mobina can benefit from the UWAT. Finally, the different components of Mobina are introduced.

3.1 Mobina application

To validate the architecture designed in this research, a case study is used to create an implementation of the UWAT and to validate the design. The SUA used in this case study is Mobina. Mobina is a content-intensive application which helps SME manufacturing companies identify what changes with a high impact on the way they work mean for their organization, processes and information landscape. Mobina positions itself as a Knowledge-as-a-Service (KaaS): their web application gives access to scarce but crucial knowledge to start implementing a high-impact change.

The content in the application is provided by the team behind Mobina consisting of experienced consultants, leadings scientists and young talent. They have a lot of knowledge on the edge of business and IT, which is scarce and therefore often expensive. By standardizing and automating this knowledge, Mobina wants to make this knowledge accessible to all manufacturing companies, including SMEs which do not often have the resources or culture to hire the alternative, namely a consultant.

To make the knowledge understandable and approachable for the manufacturing companies without human intervention, Mobina developed their own modelling technique, which led to an industry reference model. Around this knowledge in Mobina, there is an extensive collaboration environment such that employees throughout the entire manufacturing company can discuss what a (possible) change means for their processes and information landscape. By involving the people within the organization, Mobina also contributes to mobilizing the organization for the change to come.

3.2 Relevance

Mobina contains a lot of complex content and concepts. Besides that, Mobina aims at making this knowledge accessible to a whole range of people in the company, all with different background and education. To ensure everybody can grasp the content and actively participate in the process, the team behind Mobina has discussed internally, but also with potential customers, partners, and key opinion leaders, about how to structure and present the content and the software. This has led to a unique way of modelling the content, which are called reference models in Mobina, and a user interface that supports this way of modelling.

However, until now there has not been an objective measure to determine whether the content is understandable and whether the interface is practical for the end-user. User workflow analysis can potentially give the content and software developers of Mobina insights in how the user interacts with the application and which steps the user takes. This can lead to (additional) insights on how the application is used. Because the team behind Mobina already has a vision on how they believe the software is used, hence the own modelling technique, this expected behavior can be compared with the actual behavior to validate if the UWAT has an added value.

3.3 Mobina components

The content and the software of Mobina is constantly under development. This has led to a difference in functionality between when the research was started, and these data were collected, and the functionality the software has currently. This section discusses the software as when the data were collected.

Figure 5 shows the major software and knowledge components of the Mobina software. This model does not represent the actual definition in code but only helps describing the components of the Mobina software. The blue components are data that are entered by the customers of Mobina, whereas the red components represent content delivered by Mobina.

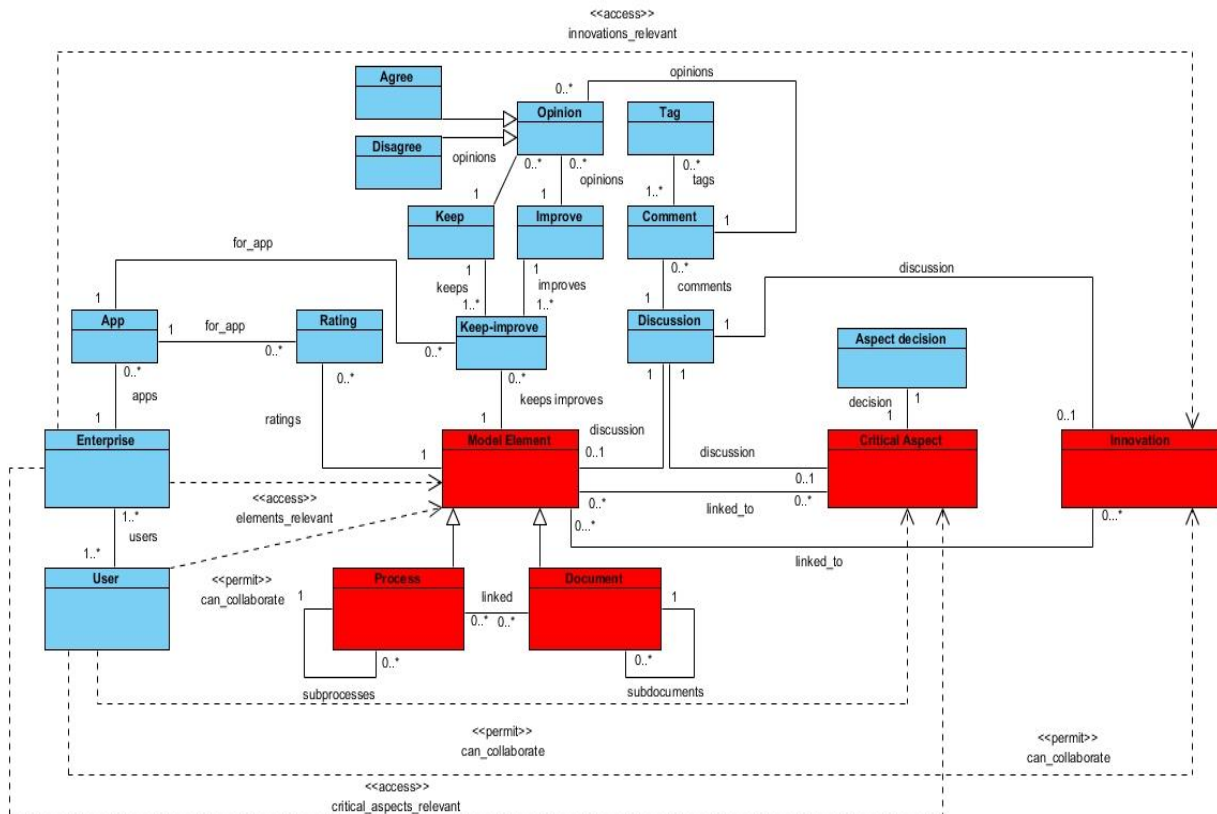


Figure 5 Software and content components of Mobina

Mobina contains three different kinds of content objects: the reference model, the critical aspects and the innovations. The reference model is the part of the software that is most used by the customers. It consists of model elements, which can be processes or documents. Mobina defines processes as actions and documents as information exchanges. Both processes and documents have a hierarchy, where each layer represents more detail. This can go up to five layers deep. Processes and documents are linked to each other. Processes of different parts of the reference model (each part represents different parts of the manufacturing company) are linked together through the documents. The assumption is that people from different parts of the organization are brought together in these documents. Figure 6 shows a screenshot of a process in the Mobina reference model.

Manufacture

This process is all about the physical goods flow. We distinguish components (or parts) manufacturing, assembly and eventually separated revision shops. We discuss how they are physically organised, how they are planned and how the processes interact with warehouse and distribution.

Here, raw materials and components are transformed into final goods or sub-assemblies that further are stored and shipped to ...

More

Process overview > PRODUCT FULFILLMENT > Manufacture

The screenshot displays the 'Manufacture' section of the Mobina reference model. It features a navigation bar with tabs: REFERENCE MODEL, INFORMATION, DISCUSSION, KEEP / IMPROVE, RATINGS, and ADMIN. Below the navigation bar is a grid of process elements, each represented by a blue card with a title and a right-pointing arrow. The 'Manufacture' card is highlighted in a darker blue. To the right of the grid is a 'Linked documents' section, which is a list of document titles with red arrows pointing to the right, indicating they are linked to the current process element.

REFERENCE MODEL	INFORMATION	DISCUSSION	KEEP / IMPROVE	RATINGS	ADMIN
Manage customer order project	Engineer product based on customer order	Plan production and release orders			
Purchase raw materials, and subcontracted and outsourced services	Manage raw materials, parts and subassemblies	Manufacture			
Store, pick and pack products	Ship and erect products	Execute outbound logistics			

Linked documents:

- Company's manufacturing systems
- Customer order related manufactured components to store
- Customer-specific product data (internal technical file)
- Finished goods
- Open purchase and production orders (scheduled receipts)
- Picked and staged raw materials and components from storage to production
- Released manufacturing (shop floor) orders

Figure 6 A process in the Mobina reference model

A collaboration environment is offered for all model elements. This consists of three major parts: discussion, keep/improve and ratings. All three parts are always linked to one model element, i.e., to one topic. In the discussion the users can place free-format comments and have an interactive discussion with each other. Figure 7 shows an example discussion. Each comment can have one or more tags. All enterprises can add for their company which applications they use, in free-format. These applications can then be tagged in comments, but are also the foundation of the keeps, improves and ratings. Using ratings, a user can give a rating between 1 and 7 and an explanation for an application. The users can also collaborate on a list of things they want to keep or improve in their applications. Since the discussion and the keeps and improves are collaborative elements, other users can give their opinion easily by saying whether they agree or disagree (similarly to thumbs up/down).

The screenshot shows an example discussion in the Mobina system. It features a vertical thread of comments. Each comment is preceded by a rating (up and down arrows) and a timestamp. The comments are displayed in a light blue box. The first comment is from 'Production Manager' and is tagged with 'Warehouse Worker', 'ERP', 'PDM', and 'Specials'. The second comment is from 'Warehouse Worker'. The third comment is from 'Sales Engineer' and is tagged with 'Purchaser Factory'. The fourth comment is from 'Sales Engineer' and is tagged with 'PDM'. The fifth comment is from 'Purchaser Factory'. The discussion is interactive, with 'Reply' buttons next to each comment.

0 | For customer-specific components, my people often need to go back into the warehouse to find missing parts. Why do we have this problem with specials and how can we solve it?
 Warehouse Worker ERP PDM Specials
 Posted by Production Manager, last edited 6 minutes ago Reply

0 | We currently use the item codes of suppliers; these are mostly not synchronized in ERP which makes it very difficult to find out where to store incoming goods.
 Posted by Warehouse Worker, last edited 5 minutes ago Reply


4 | I see two options to solve this. On the one hand, we could manually enter the article codes for specials in ERP or we need a plugin that can do it automatically for us. On the other hand, for regular orders our suppliers print our article codes on the delivered goods. I believe these article codes are defined by engineering. Maybe we can do this for customer-specific components as well?
 Sales Engineer
 Posted by Purchaser Factory, last edited 5 minutes ago Reply

2 | Currently we don't generate article codes for customer-specific components since they're often not produced on our specification and we don't have a structural purchasing contract. If you think the suppliers will do it the same way, then we can easily set it up in our PDM system which is exported to ERP.
 PDM
 Posted by Sales Engineer, last edited 5 minutes ago Reply


0 | For most goods we order we have longer term agreements with suppliers, which will make it easier to also agree on this for one-time products. Just a small amount of goods is bought outside of these agreements.
 Posted by Purchaser Factory, last edited 4 minutes ago Reply

Figure 7 Example discussion in Mobina

The second type of content objects is critical aspects. These are complex business aspects that have a deep impact on the IT landscape of a company. It is often hard for a company to oversee the impact of these aspects on IT applications, which is why Mobina defined them from a business perspective. For each of these aspects there is again a discussion functionality, which is similar to the discussion of the model elements. The critical aspects are also linked to the relevant model elements to enable the discussion in a broader perspective. For each of these aspects, the users can also determine the priority of the critical aspect now and in the future, and the impact on their IT landscape. In this way, their impact is discussed, and the company establishes an agenda for a futureproof IT landscape. Figure 8 shows an example of the priority and impact given for a critical aspect.

 The interaction between the bill of material (BOM) and routing


The interaction between the BOM and routing is an important aspect in every information system surrounding the production floor. The focus is often on one or the other, influenced for example by the production system and the nature of the production. It is important to set up your information system in a way that it has the same starting point ...

More 

DISCUSSION
INFORMATION
PRIORITY & IMPACT

Priority


How much priority would you give to this critical aspect **now**?



Explain here why you give this priority

Currently we use a more traditional assembly method with only focus on the BOM. Therefore, the interaction between BOM and routing is not really a hot topic, since the experience of our people ensures that they know themselves which production steps to take.

How much priority would you give to this critical aspect **in the future**?




Explain here why you give this priority


The decreasing availability of skilled production employees will probably force us to standardize our production processes more and more. Therefore, having a well linked BOM and routing will become increasingly important and we should already take this into account in major decisions for our information landscape.

CANCEL SAVE

Impact

 Find an app

ERP - manufacturing

Very high impact 

Explain here why you give this impact

Our production module delivers the most important information to the shop floor and is thus highly impacted by any changes.

REMOVE SAVE

Figure 8 Priority and impact for a critical aspect in Mobina

The last type of content object is innovation. Innovations in Mobina are the well-known innovations in the manufacturing industry like Internet of Things, Smart Manufacturing etc. The role of innovations in Mobina is to include them in the entire discussion and assess the impact of the innovations on the processes and information landscape of the company. The innovations are therefore linked to model elements. For each innovation, there is again a discussion space, which is the same as for the model elements and the critical aspects.

Before the company starts using the software, Mobina asks the contact person some questions to configure the software. Based on these questions the typology of the company can be made and the relevant parts of the reference model, as well as the relevant critical aspects and innovations can be determined. The company also has possibilities to restrict the user's permissions. Figure 5 shows that user access to model elements, aspects and innovations can be restricted. The configuration of the content and the permission system can be relevant for the user workflow analysis, since not all functionality and content will be available for all users.

4 Solution

This chapter presents our solution. The chapter starts with the functional, non-functional and domain requirements for the UWAT architecture. Then the high-level architecture and its external interfaces are discussed. Finally, we discuss the different techniques selected to implement the UWAT and their combination.

4.1 Requirements

The requirements for the UWAT architecture, referred to as system in the requirements, are divided into three categories (according to the division of Sommerville [35]): the functional requirements, non-functional requirements, and domain requirements.

These requirements reflect what is needed to perform user workflow analysis on content-intensive applications and make an architecture that enables all different SUA owners to achieve their objectives. They do not represent what the SUA owner would want to achieve with this architecture.

4.1.1 Functional requirements

Functional requirements are defined by Sommerville as follows: “statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations”. For this research, the following functional requirements are defined:

1. The system should be able to extract the specified patterns from the input data of the system.
2. The system should be able to give users insights in the data that cannot be directly obtained from the data.
3. The system should be able to process the data in a deterministic way.
4. The system should be able to handle different versions of the input data.
5. The system should be able to provide the results specified by the users.
6. The system should be able to use the information about the content-intensive application in its analysis.
7. The system should return a data set with the outcomes of the analyses.

4.1.2 Non-functional requirements

In this report, the definition by Kotonya and Sommerville is used for non-functional requirements: “Requirements which are not specifically concerned with the functionality of a system. They place restrictions on the product being developed and the development process, and they specify external constraints that the product must meet” [36]. For this research, the following non-functional requirements are defined:

8. The system should need no user interaction after the specification of the analysis.
9. The system should have as much as possible generic analyses that are applicable to all content-intensive applications.
10. The system should be flexible.
 - 10.1. The system should be easily extendable with new analyses.
 - 10.2. The system should be easily made applicable to new functionality of the product that is analyzed.
11. The system needs to be presented to and used by the user as one system even if it is an implementation containing multiple tools and techniques.
12. The data integrity should remain when data needs to be converted to another format within the system.
13. The user should be able to use the system and understand its interfaces without needing to know how, and with what techniques, the system is implemented.

14. The system should be able to perform the analysis within reasonable time².

4.1.3 Domain requirements

Domain requirements are defined by Sommerville as follows: "Requirements that come from the application domain of the system and that reflect characteristics of that domain". In this research, the requirements related to the input and output data of the architecture to be developed are part of the domain requirements.

15. The input data should have timestamps.

16. The output data of the system should be a standardized format that can be used by different visualization tools.

4.2 High-level architecture

Figure 9 shows a component diagram which represents the UWAT and its interfaces to the external environment. The UWAT implements the user workflow analysis. It processes input data that it receives from the user preparation interface and has as an output the information for the result dashboard. This component is implemented using MDE. The next chapter discusses the different models and transformations.

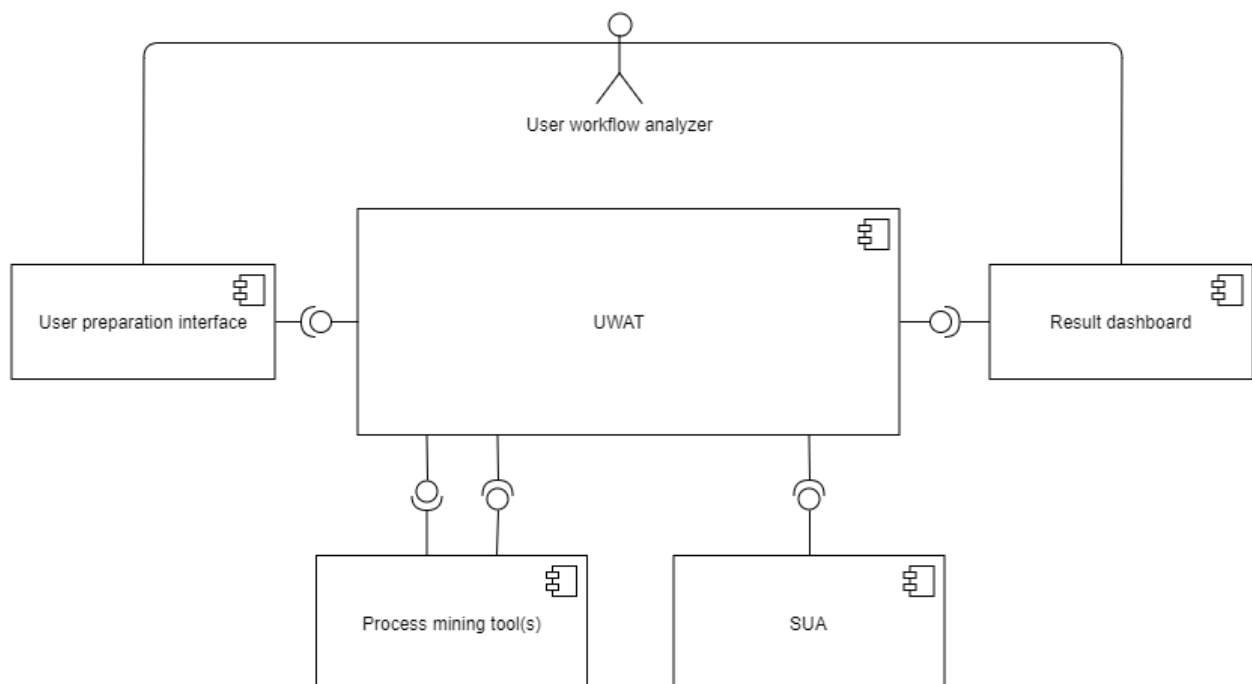


Figure 9 High-level architecture

The *user preparation interface* is the interface in which the user workflow analyzer can decide what they want to analyze, as well as setting some possible parameters for this analysis. The options, parameters and data sets are defined by the SUA owner and are predefined and implemented before a user workflow analyzer starts analyzing the SUA. The selected data set and options are passed to the UWAT. The *result dashboard* is the interface to the user workflow analyzer to present the outcomes of the analysis. These outcomes are passed from the UWAT to the dashboard. For the user workflow analyzer these two interfaces may be the same, but they represent completely different views and are therefore modelled separately here.

² Reasonable time depends on the requirements of the SUA and how the UWAT is used. There is no standard answer here.

The user preparation interface and result dashboard are intentionally separated from the implementation of the UWAT. The implementation is the responsibility of the SUA owner. In this way, the SUA owner has the flexibility to determine how to visualize the options and results. It also gives the options for extra pre- and post-processing.

There are three other external interfaces to the UWAT. Two of them are with one or multiple PM tools. The implementation of PM is not part of the UWAT. There are many (open source) tools available which can deliver the requested PM results. By keeping PM separate, the user can benefit from research and developments in these tools. The implementation of PM is treated as a black box here. There are only two interfaces: one to specify the results and one to extract the outcome of the PM plugins.

The final interface is with the SUA. This interface represents the information that is extracted from the SUA and is used in the user workflow analysis. As discussed in Chapter 1, there is no direct link between them. However, there is an information exchange which is represented by the interface in Figure 9. This information exchange includes the SUA data and the log data. We assume the log data will be passed to the PM tools via the UWAT.

4.3 Techniques

In this architecture, the techniques PM and MDE are used. PM is used to extract the user workflow from the event log data. As discussed in Section 2.2.2, for this type of analysis the strength of PM is discovering processes and analyzing time series. However, there is no standardized way yet to analyze the results of the PM plugins, i.e., the different PM algorithms and techniques; this is left up to the user.

MDE is introduced in this project to bridge the gap from the PM results to results of interest for the user workflow analyzers, because of its characteristics described in Section 2.1.3. MDE is used to configure the PM and interpret the results in a standardized way. Since MDE abstracts certain concepts, it has the potential advantage to implement (part of) this design for multiple SUAs, i.e., not specific solutions for every single SUA. Its main drawback is that it mainly works if there is a small DSL. This will probably not impose any limitations, since the design only focuses on the user workflow analysis of content-intensive applications, which is already a narrow domain. Another drawback is that the tools are often limited to one platform. Since this design presents a stand-alone tool this is also not expected to impose a limitation.

In this project, PM is embedded in an MDE approach to provide a flexible, automated and standardized way of executing PM and interpreting the results. The potential user workflow analyzer does not need to understand the PM plugins and results and analyze what the results mean for themselves. The user workflow analyzers also do not need to know any implementation details of PM. No references have been found that MDE is used to abstract from the PM execution before this research.

5 (Meta)models and transformations

This chapter presents the different metamodels and transformations used to implement the UWAT. Firstly, the transformation chain is presented to give an overview of the implementation. Next, the five metamodels are introduced, including an illustrative example of the case study. Finally, the four transformations are discussed. For the applicable transformations, also the format of the input and output data of the UWAT is introduced.

5.1 Overview

Figure 10 shows an overview of the transformation chain used in our solution. This figure only contains the design details related to MDE (i.e., the models, metamodels and transformations). The blue elements represent external information or tools. The orange elements represent the data exchanged through the interfaces with the user workflow analyzer.

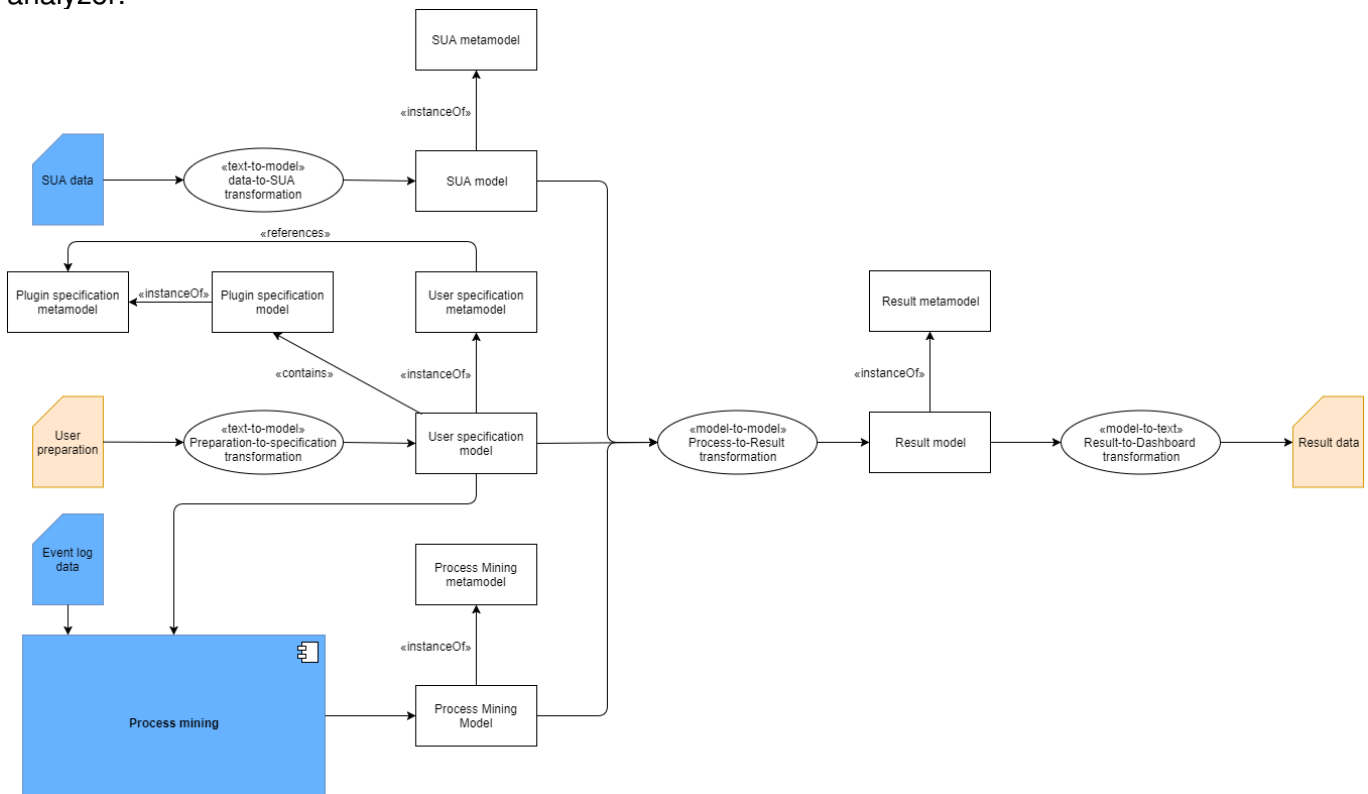


Figure 10 Transformation chain

5.2 Metamodels

To implement the transformation chain, we defined the following five metamodels: SUA metamodel, plugin specification metamodel, user specification metamodel, Process Mining metamodel, and result metamodel. We defined all metamodels as generic as possible, so it could easily be applied to different SUAs. For each metamodel, a simple example from the case study is presented to illustrate the models that instantiate the metamodel.

5.2.1 SUA metamodel

The SUA metamodel represents the information of the SUA that is relevant for the user workflow analysis. The architecture is developed in such a way that as little as possible information about the SUA is needed. This makes it easier to apply the same architecture to different SUAs. Therefore, we decided to put all information about the application in a separate metamodel. When the architecture is used for a different SUA, a different SUA metamodel must be defined.

We illustrate the SUA metamodel with the metamodel created for the case study in Figure 11. The main content class represents two attributes that are relevant for every SUA: the name of the application and the version. The name of the application specifies the SUA and can be an ID or another discriminating attribute that the SUA owner uses. The version is added to differentiate between different version of the SUA (or content versions if applicable). The other classes are SUA specific. For the case study, the content elements were needed to include the structure and the names in the result dashboard.

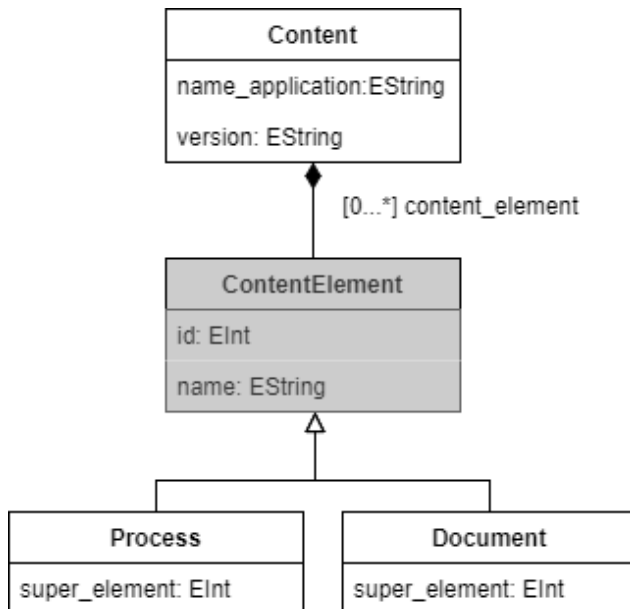


Figure 11 Mobina SUA metamodel

Example case study

In the case study, the SUA metamodel is used to store the names and hierarchy of the reference model in the Mobina software. The model instance of this metamodel containing one process and one document looks like:

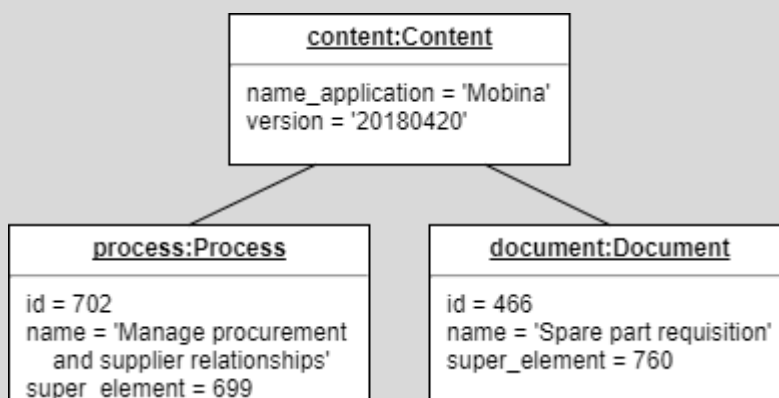


Figure 12 SUA model example

5.2.2 Plugin specification metamodel

The PM execution is treated as a black box in the architecture. External tools are used to execute the PM algorithms. However, to establish what analyses are possible, one must know what algorithms are available and the input and output of these algorithms. Therefore, we introduced the plugin specification metamodel. An instance of this metamodel represents a plugin of a PM tool.

Figure 13 shows the plugin specification metamodel. The metamodel defines a generic way to model plugins. An advantage of this generic setup is that it can be applied to different tools and plugins. This also raises the opportunity to share the specification among different SUA owners. In the plugin the tool and the name of the plugin are specified. The plugin has zero or more parameters, which are also generically defined to make them tool and plugin independent.

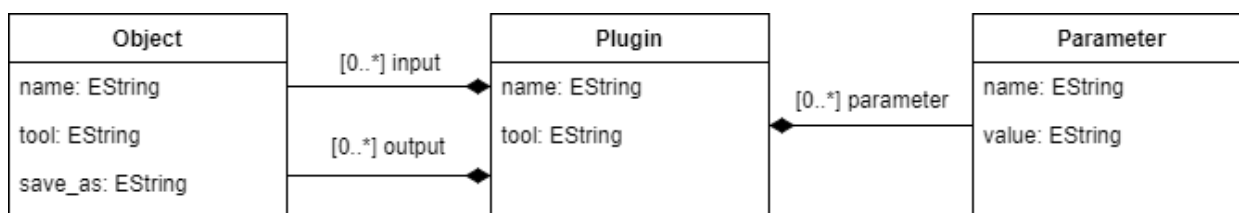


Figure 13 Plugin specification metamodel

Each plugin can also have zero or more input objects and zero or more output objects. This represents the information that must go into the plugin, and the different outputs. The outputs of one plugin can then be the input of another plugin. These can be from different tools, so next to the name of the objects also the tool is defined. The last attribute is `save_as`. In this attribute one can find the name of the file, which can be used to find the correct result file for the rest of the execution.

Example case study

In the case study, three plugins are defined in compliance with the plugin specification metamodel. One of the plugins is the plugin 'Add time between events (Duration) as Attribute to all Events' of the tool ProM. The input is the log data and the output a transformed log file. There is one parameter, which represents a new attribute name. For this plugin the plugin model instance looks like:

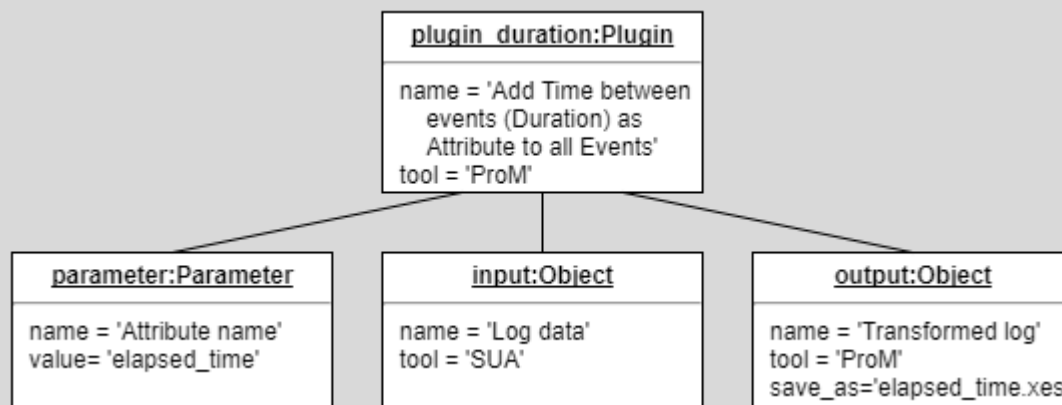


Figure 14 Plugin specification model example

5.2.3 User specification metamodel

The user specifies what he/she wants to analyze through an interface. This specification is then passed to the UWAT, which processes and prepares these data, so it can be used for PM and the rest of the analysis. What the user wants to analyze is defined in the user specification metamodel.

Figure 15 shows the user specification metamodel. All classes are generic except for the definition of the datatypes, which is an enumeration. The architecture assumes that every analysis analyzes one data set at the time. This was decided to create a clear result for the end user. However, the model can easily be adapted to support multiple data sets in one analysis, when this is deemed beneficial. When the architecture is applied to a different SUA, the SUA owner defines the types of data sets available. In the case study, we could evaluate the model elements (i.e., the processes and documents) or the different tabs (i.e., different functionalities at a model element).

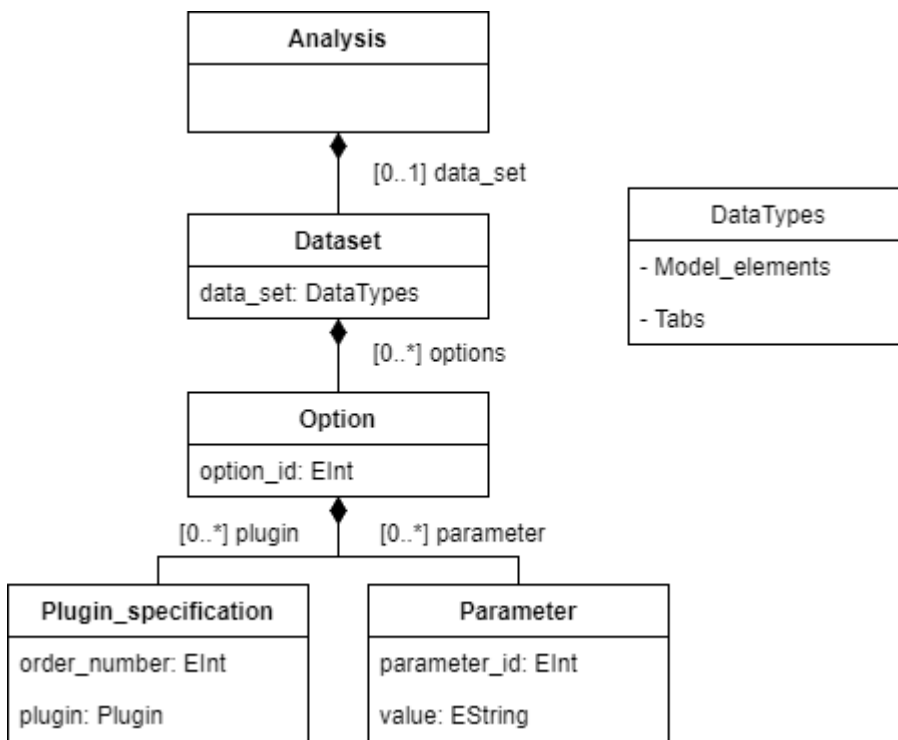


Figure 15 User specification metamodel

For each dataset options are available. These options are predefined by the SUA owner when the architecture is implemented. In other words, the SUA owner must determine what could be analyzed by the user workflow analyzer. This should be based on the business objectives of the SUA. For each option, the SUA owner can also define a set of parameters (e.g., how many results should be returned). For each option, a set of plugins should be selected. The plugins are instances of the plugin specification model. To each selected plugin an order number is added. Based on the order numbers and the information from the plugin specification models, the PM instructions can be defined for the tool to execute them. The implementation of these instructions depends on the external PM tool. It is up to the implementation of the PM to ensure these plugins can work together.

Due to the split in the plugin specification and the linking of these plugins to specific analyses, this design raises the opportunity to define the plugins already with no particular analysis in mind or to easily reuse the plugin specification for multiple analyses.

This metamodel contains a lot of flexibility, because we want the architecture to be flexible. This way different kind of requirements and analyses that SUA owners might have are supported. However, this also means that this model and its implementation need to be discussed carefully when the architecture is implemented, to make sure the user workflow analyzer can effectively execute his task.

Example case study

In the case study there is a data set with the data type 'Model_elements'. For this data set three options are defined. For the first option, there are two parameters, which are identified by their parameter id. One PM plugin is specified for this option, the example of the previous metamodel. The model instance for this option is:

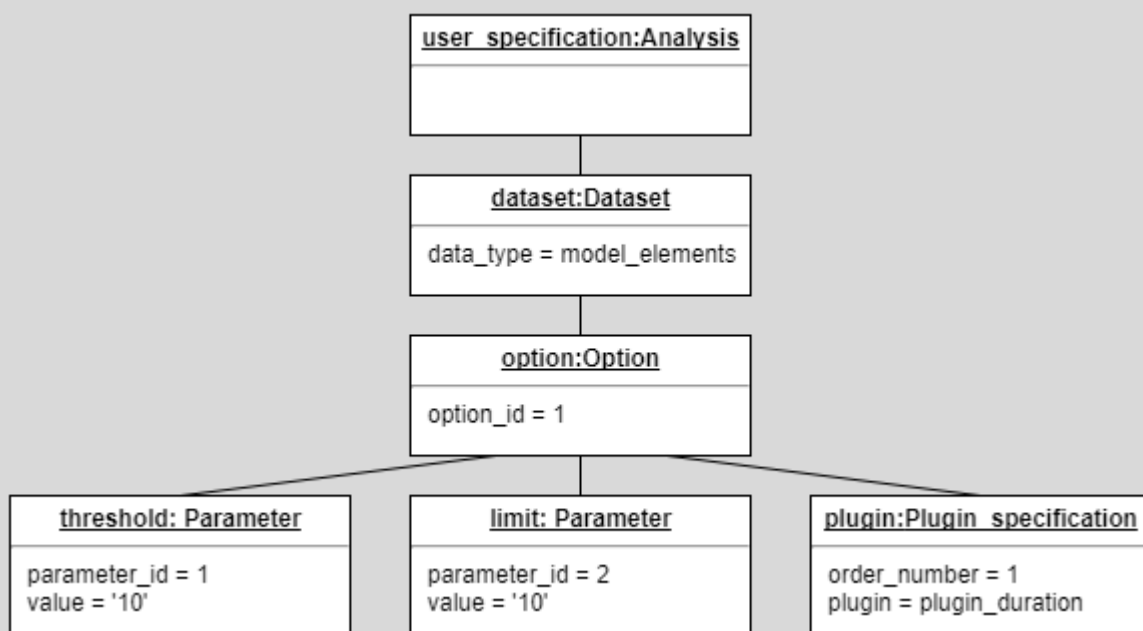


Figure 16 User specification model example

5.2.4 Process Mining metamodel

The PM metamodel is designed in a way that it captures the relevant results of the PM techniques and algorithms, but also generic enough to contain no implementation details of specific plugins. The PM metamodel presented in Figure 17 models two important result concepts of PM: analyzing the event log data and discovering a process model. This model can be extended with other PM concepts (d, conformance checking). The PM metamodel contains the combined results of all PM executions and is not limited to one plugin execution.

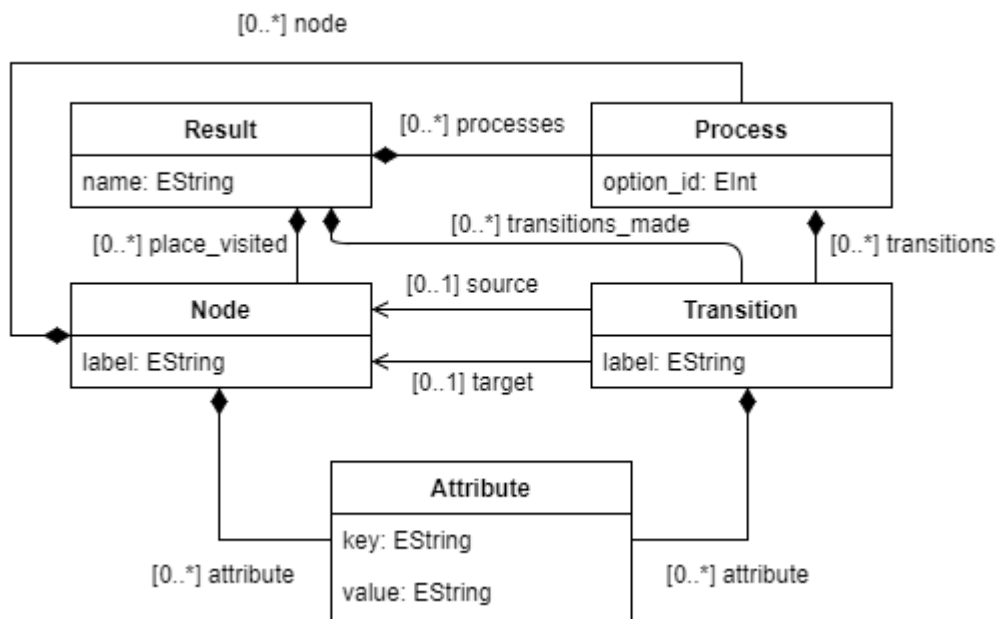


Figure 17 Process Mining metamodel

The two basic elements of PM results are nodes and transitions. Nodes are the different places visited, for example, the different processes and documents in the case study. The transitions are the movements from one node to another. Nodes and transitions both have a label to represent them in a generic way. Based on the context this can be processed later in the process-to-result transformation. All other attributes for nodes and transitions (e.g., frequency) are modelled in the same flexible way as the parameters in the user specification metamodel.

In the metamodel, two potential outcomes of PM are modelled. First, the process class in the metamodels represents a discovered process by a PM plugin. This is set of nodes and transitions, which are linked to the process instance. Each option can have a discovered process; they are differentiated by the option id. One can argue that the nodes are linked to the process through the transitions with a discovered process. However, some discovered processes have isolated nodes. Therefore, the nodes are also linked directly to the discovered processes.

The second potential outcome is the enhancement of the event log or complete set of nodes and/or transitions. In this case, the nodes and transitions are enhanced with extra attributes. Because these enhanced nodes or transitions are not part of a (discovered) process, they are directly linked to the result instance via the place visited or transition made relationship.

Process has an attribute option id, which is the same id used in the user specification model. Because of this shared option id, the correct outcome can be linked to the corresponding analysis. This is not necessary with directly linked nodes and transitions to the result, since these contain option results that create no different visualization or representation but are merely attributes added to these data. The option can be identified based on the attribute being available and therefore the option ids do not need to be stored separately in the PM model.

Example case study

The PM executions for the case study created one process discovered by a process discovery plugin. For all other options the nodes added via the place_visited relation are used. For the option used in the example of the user specification metamodel, the place_visited instances are used with the attribute frequency. The PM model instance for one node and transition in the discovered process, and one place visited looks like:

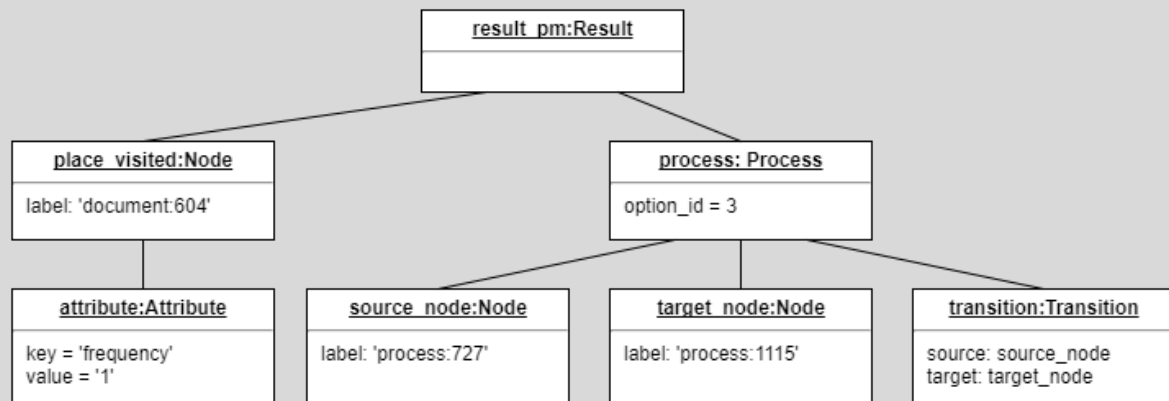


Figure 18 PM model example

5.2.5 Result metamodel

The result metamodel is a direct representation of the results for the user workflow analyzer. In other words, the user specification and the PM outcomes are already translated here to the results the user workflow analyzer specified. The result-to-dashboard transformation only needs to print these results. A great advantage of this set-up is that only once a visualization needs to be created for each result type. When a new option has the same result type as previously defined options, the same visualization can be used.

Figure 19 shows the result metamodel. The result is for the data set as defined in the user specification. Since this is only used here for printing it in the result, the data set is now a String instead of a value from the enumeration. The result can be two types: a list or trace collection. The list is simply a set of elements. A trace collection is a set of traces. A trace is a source with zero or more targets. In other words, a list only shows the elements whereas trace collections also show relations between these elements.

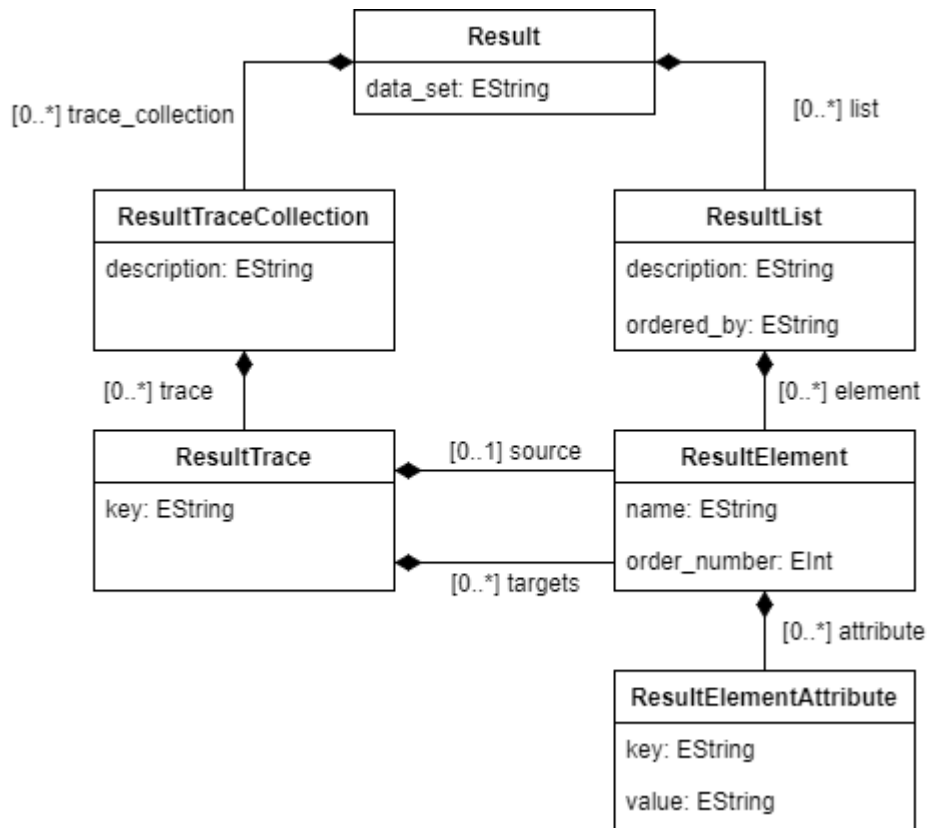


Figure 19 Result metamodel

Both lists and traces use result elements. Each result element has a name and an order number if the ordering of the result elements is relevant. To keep the result as generic as possible, the attributes are again shown as a separate class, where each attribute has a key and a value.

This model represents generic result types, i.e., list or trace collection. This can be easily extended with other result types. In this case, the generic class result element is still useful since all kind of elements can be represented like this. The metamodel contains no details anymore about the data set and the different options or parameters. This is only represented by the strings data set and description, so it can be visualized. However, this metamodel is completely separated from any knowledge about the system and the specification.

Example case study

In the case study there are two lists and one trace collection as a result when all options are selected. The result model for one trace in the trace collection and one list item for the options used in previous examples looks like:

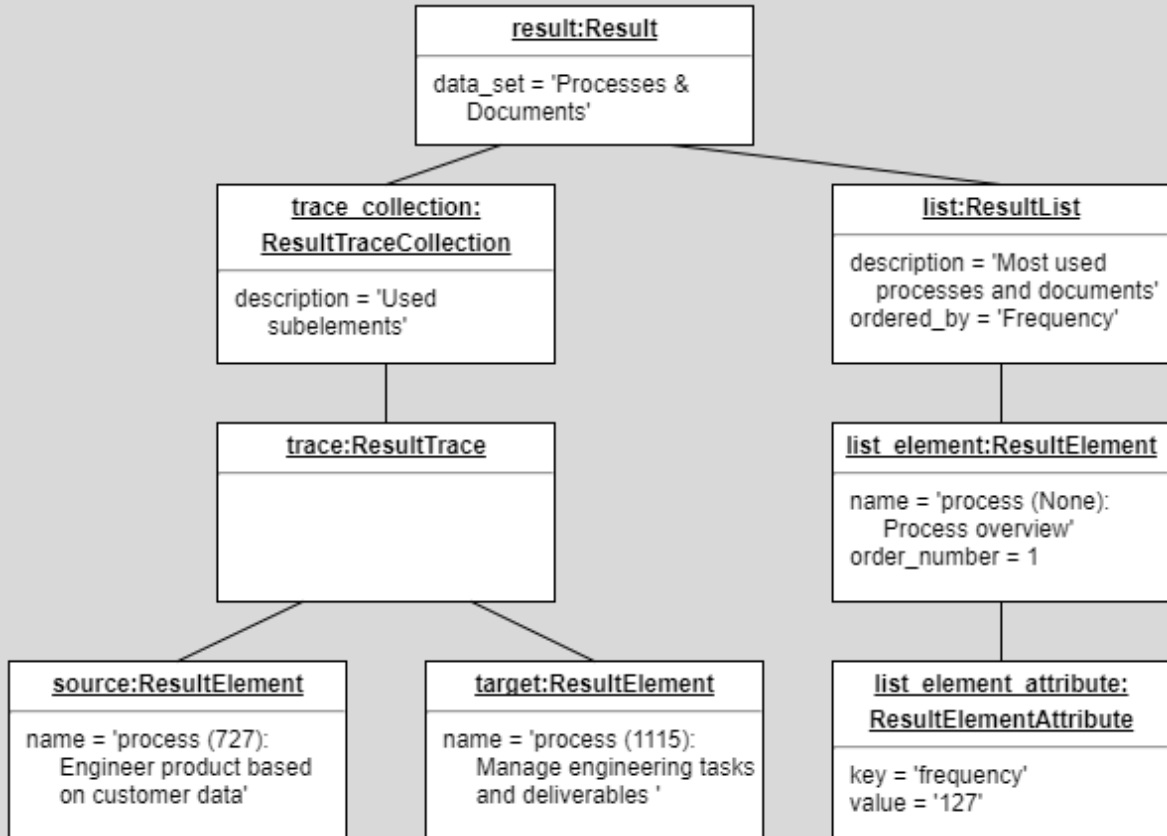


Figure 20 result model example

5.3 Transformations

The following transformations need to be implemented in the transformation chain.

5.3.1 Data-to-SUA transformation

This text-to-model transformation transforms the data of the SUA to a model instance of the SUA metamodel. Since the metamodel is specifically made for the SUA, also this transformation needs to be developed specifically for each SUA. What should be in this transformation and how this input is delivered is up to the SUA owner and there are no specific constraints on this.

5.3.2 Preparation-to-specification transformation

The input for this text-to-model-transformation is the input of the user workflow analyzer in the user preparation interface. The input needs to be a JSON dictionary object according to the following format:

```
{
  'option_id': string,
  'parameters': [
    {
      'parameter_id': string,
      'parameter_value': string
    }
  ]
}
```

In the JSON dictionary object, for each key the value type is defined. In the case of a list, the type of the list items is within the list. There can be zero or more items in the list; only the type is defined here.

This transformation translates the input of the user workflow analyzer to an instance of the user specification metamodel. The different data sets available and the different options and related parameters available are already defined by the SUA owner upfront; the user only has these options. In the implementation of the user preparation interface, only the selected options must be added to the JSON object, i.e., if an option is in the input dictionary, this transformation knows that analysis must be performed.

Because the user specification model contains the plugin specification models, this transformation should also know which options relate to which plugins. This transformation can translate the selected option and data set combination to a set of PM plugins.

This transformation is standard for different SUAs in a sense that the input data format can be standardized and therefore the options and parameters can be extracted from the input automatically. However, the SUA owner must extend this transformation with the relevant plugin specification models, i.e., link the different options to the plugin specification models.

5.3.3 Process-to-result transformation

This model-to-model transformation contains all the logic of the user workflow analysis and therefore there are a lot of specifics for the SUA. The input for this transformation is a PM model, a SUA model, and a user specification model. The output is a result model. This means that this transformation must know which options and parameters give which PM results and what the PM results mean. Besides the definition of the inputs and outputs, this transformation can therefore not be generic for multiple SUAs.

This has the disadvantage that the implementer of the architecture still needs a lot of knowledge about MDE and needs to understand the models. However, except for the SUA information and the data set and option definitions, all changes per SUA are confined to this transformation. All other metamodels and transformations could therefore be made generic and easily extendable to other SUAs. Due to the metamodels definition, the implementer knows what to do here. It only needs the SUA specific knowledge to implement it.

5.3.4 Result-to-dashboard transformation

This model-to-text transformation has as a sole purpose to print the results of the analysis. Because the result types are already defined in the metamodel and because these are generic, this transformation is the same for each SUA. Of course, each SUA could use the result of this transformation in their own environment to give it their look and feel, but that does not change the data output of the analysis, i.e., this transformation.

The output of this transformation is again a JSON dictionary object. This output object is very similar to the result metamodel. It has the following structure (the same notation is used as with the input JSON object):

```
{
  'data_set': string,
  'lists': [
    {
      'description': string,
      'ordered_by': string,
      'elements': [
        {
          'name': string,
          'order_number': int,
          'attributes': [
            {
              'key': string,
              'value': string
            }
          ]
        }
      ]
    }
  ],
  'trace_collections': [
    {
      'description': string,
      'traces': [
        {
          'source': string,
          'targets': [string]
        }
      ]
    }
  ]
}
```

6 Implementation prototype

To validate the architecture, a prototype was implemented for the case study. This chapter discusses the implementation of the prototype. First, the scope is set, and we present the options and parameters that are implemented. Then, the different tools and techniques are introduced. This is followed up by the actual implementation, which is split up in four parts. First, the preparation that should be done upfront is discussed. Next, the user preparation interface and the generation of the input data is showed. Then, the implementation of the UWAT and PM is presented. Finally, the presentation of the results in the result dashboard are presented for this prototype. Where useful, the implementation is illustrated with (pseudo)code or screenshots.

6.1 Scope

Before an SUA owner starts implementing a UWAT, first a list of business objectives should be defined, to determine what the SUA owner wants to achieve with the UWAT. The complete list for Mobina can be found in Appendix A. The list does not need to be implemented completely to validate the design. Three business objectives have been implemented in the prototype. To limit the scope, these are limited to questions about the content (not the software) and to the reference model (not the critical aspects and innovations). These objectives are selected based on their added value for Mobina and the expected benefit from user workflow analysis. The three selected objectives are³:

1. Do they come back to the same place? (1d)
2. How long do they stay at the same process or document? (1c)
3. What subprocesses or subdocuments are used? (5c)

The next step is to translate these business objectives to options and useful parameters for the user workflow analyzers. We selected the following three options and parameters:

1. Most used processes and documents
 - a. Minimum times used
 - b. Limit result to top x results
2. Most time spent at processes and documents
 - a. Minimum time spent (time in seconds)
 - b. Limit result to top x results
3. Subelements used
 - a. Only direct relations (boolean)

The first option can answer the first objective, the second option the second objective and the third option the third objective. For objective 1 and 2, the parameters are a threshold and the number of results to be returned. These parameters are added to give the user workflow analyzer some flexibility in what they interpret as useful results. In a complete implementation this may be extended with more options; these options are only included to illustrate the purpose and role of the parameters. For the last objective, the parameter illustrates a typical parameter specific for Mobina. The reference model is a complex structure, which is often difficult to understand. Parameters like this illustrate how this analysis may give added value for an SUA. For now, only direct relations are implemented.

6.2 Tools

To implement the complete prototype, different tools and libraries are used. For the entire development, the Mobina environment is used. In the backend, Mobina uses Django, which is a Python framework. The call to the different transformations for example, happens within this backend. For the frontend, Mobina uses VueJS as a JavaScript framework and

³ Between parentheses the number in the original list of business objectives is given

MaterializeCSS as a CSS framework. The user preparation interface and result dashboard were developed in this environment with these frontend frameworks.

The UWAT itself is implemented using MDE. Eclipse Modeling Framework (EMF) is the most popular open source MDE framework. EMF is a modelling framework and code generation facility that enables building applications based on a structured data model⁴. EMF is developed in Java, and the output files are also in Java. The core (meta)model of EMF is Ecore, which functions as a metamodel.

In this research, PyEcore is used. PyEcore⁵ is an open source Python EMF implementation. It has almost the same functionality as the Java implementation. PyEcore is used because we are more familiar with Python and this also enables the implementation of the UWAT to be embedded in the development environment of Mobina. For the metamodels, the Ecore definition of the EMF framework is used. The different models are XMI models compliant with the Ecore metamodel. The transformations are implemented as Python scripts. The PyEcore library can load and register the different metamodels and load the different XMI models which makes it possible to use them in the different transformations. When applicable, the PyEcore library can also serialize the objects from the transformations to XMI models so they can be loaded in another transformation.

For PM, the external tool ProM is used. ProM, short for Process Mining Framework, is an Open Source framework for PM algorithms, developed by the Eindhoven University of Technology. Their vision is to actively advance the state-of-the-art of PM technology by developing methods that really work, by creating an open community, and by providing a stable and easily extensible platform, which optimally supports PM [37]. We used ProM in this research for two reasons. Firstly, because it is open source we can use all functionality for this research and if the research has a positive outcome the implementation can also be used commercially. Secondly, ProM is continuously extended with new functionality and improved algorithms. In this way, the newest and most up-to-date functionality and algorithms available can be used.

6.3 Preparation

The SUA owner needs to prepare two things before the UWAT can be used: the event log data must be prepared so it can be used for PM and the application model must be created. This section discusses how these two actions have been executed for the case study.

6.3.1 Preparing the event log

For the case study, the log data is available as a JSON export from the database. ProM however expects an event log in the XES format⁶. Besides that, the event log does not necessarily need to include all activities. Therefore, an extra Python script was written to convert this JSON export to a XES event log which only contains the relevant events.

When creating an event log, one needs to think about what a trace should be and what event information is needed. In our prototype, we use the data set processes and documents, where each event is a switch to a new process or document, together called model elements. A trace in our data set is a session of a user, i.e., all the actions a user performs from login to logout or closing the tab.

⁴ <http://www.eclipse.org/modeling/emf/>

⁵ <https://pyecore.readthedocs.io>

⁶ <http://xes-standard.org/>

In XES, the structure of the event data (and potentially the traces) is defined upfront in an XML format using a *global*. This can contain the standard attributes of XES but also self-defined classifiers. *Classifiers* are attributes that can later be used to filter result models, or these can be used as parameters for some plugins. Figure 21 shows the global event definition and classifiers for the used event log. For each event, we used the standard attributes name, resource and time. Resource, who or what executed the action, was added for future reference but is not used currently. The name is in the format TYPE:ID where type is process or document and id is the id of the model element. Since this combination of type and id is unique, the element can be unambiguously identified in the rest of the analysis. The timestamp is the moment the switch to that model element happened.

The other four attributes are classifiers. *Source* is how they got to this element (i.e., via the directory, via a process card, etc.). *Type* is whether it is a process or document. *Id* is the id of the model element. *Sub* is whether at the point of action the user viewed the sub elements or the siblings. *Type* and *Id* are already in the model element name but adding these as classifiers separately allows filtering on only one of these attributes. The final two classifiers are combinations of the other classifiers.

The event log contains more information that is needed for these three analyses. In this way, all other relevant analyses as defined in the business objectives can also reuse the same log data.

```
<global scope="event">
  <string key="concept:name" value="name" />
  <string key="org:resource" value="resource" />
  <date key="time:timestamp" value="2018-01-01T00:00:00.000Z" />
  <string key="Source" value="" />
  <string key="Type" value="" />
  <string key="Id" value="" />
  <string key="Sub" value="false" />
</global>
<classifier name="Source" keys="Source"/>
<classifier name="Type" keys="Type"/>
<classifier name="Id" keys="Id"/>
<classifier name="Sub" keys="Sub"/>
<classifier name="source classifier" keys="Type Id"/>
<classifier name="source type classifier" keys="Source Type"/>
```

Figure 21 XES global and classifiers

6.3.2 Data-to-SUA

Creating the SUA model is about creating an XMI model which contains all relevant information of the SUA that is needed for the analysis. The SUA metamodel for this case study is already included in Section 5.2.1. In this implementation, this is about converting some of the data about the reference models from the Mobina database to an XMI model.

To implement this, we make an export of the Mobina database in JSON, just as for the event log data. We only needed the processes and documents here, and not all the other content elements from the Mobina database. The script loops through all the exported content elements. When an entry is a process or document, a content element as defined in the SUA metamodel is created with the id, the name and the super element. Special entries were created for the process and document overview, since these are relevant elements that show up in the analysis but are not elements in the Mobina database. The overviews have null as id.

Finally, a content object is created, where the version is the date since Mobina does not define versions yet. The name of the application is Mobina. All the content elements are added to this content object. At the end, these content objects are serialized to an XMI model and can be used in the process-to-result transformation.

In this case study, no new reference model is uploaded during the time of data collection, so creating the model once is enough. However, in the case of a continuous data collection over different content versions, new versions of the reference model must be constantly updated, and the version of the log data must be aligned with the version of the content. This was out of scope for the case study.

6.4 User preparation

The UWAT expects as an input a JSON object which contains the specification of the user workflow analyzer. How this JSON object is created is left up to the SUA owner. In this prototype, as would be expected in many of the implementations, the user workflow analyzer specifies the results in the environment of the SUA. This environment is familiar for the user workflow analyzer and the content that is analyzed is easily available, which is expected to lead to a more user-friendly solution for the user workflow analyzer. Another added value is that the implementer can use the development environment and frameworks he is familiar with, which is expected to lead to a faster implementation.

In the implementation of the prototype, the user workflow analyzer is first asked to select a data set. After a data set is selected, the user can select the options. These are the specific options for that data set. Figure 22 shows the data set and options for this prototype.

Figure 22 User specification interface after the data set is selected

When an option is selected, the parameters for the options show up. Figure 23 shows the interface after all the options are selected and the parameters are entered.

Figure 23 User specification interface after the options are selected

Using the VueJS implementation one can define a dictionary which can contain the information about the fields, as well as contain the actual values that are currently in the input fields. These attributes can then be easily used and changed within the visual frontend for the user workflow analyzer. As an example, the definition of option 1 can be found in Figure 24.

The option id and parameter id correspond to the option and parameter ids used in the UWAT. This creates consistency through all implementations. The label for the option and the parameters is the text shown to the user workflow analyzer to describe the option/parameter. For parameters, the type is used for the validation.

The value for the parameters is the current value of the parameter. The amount shown in the figure is the default value. When a new value is entered, this field will contain the current value. This is the same for the attribute checked, which contains whether the option is selected or not. For the thresholds, the default value is set to 0 since the results should not be unnecessarily influenced. For the limitations, the default value is set to 10 since this is a convenient number. The checkbox for only direct relations is set to true by default, since this is expected to be most used. These default values are only guidelines and every SUA owner can decide for themselves which values to use.

```
this.options = [{
  option_id: 1,
  label: 'Most used processes and documents',
  parameters: [
    {
      'parameter_id': 1,
      'type': 'number',
      'label': 'Minimum times used',
      'value': 0
    },
    {
      'parameter_id': 2,
      'type': 'number',
      'label': 'Limit result to top x results',
      'value': 10
    }
  ],
  checked: false
}, {
```

Figure 24 Dictionary to define options and parameters

After the analysis is started, the data from this dictionary is translated to a JSON dictionary object containing only the relevant attributes and the selected options. Since the analysis now runs within seconds, the interface works synchronously and gives a loading sign until the results are back. Within this method, an AJAX call is done to start the analysis. With this AJAX call, the JSON dictionary object is sent to the backend.

6.5 UWAT and PM implementation

In the design, the user does a call to the UWAT which is responsible for handling the user workflow analysis. In the implementation of the prototype, this automated workflow is implemented using an AJAX call that sends a POST request. The back-end then calls the different MDE transformations and methods relevant for the PM execution. Usually the PM implementation will be external. However, this was not yet available and therefore this prototype also contains part of the PM implementation.

Figure 25 shows the POST method. The POST method results in the sequential execution of five different methods. Three of these methods are MDE transformations. The five methods are:

- *Create_user_specification_model* is the preparation-to-specification transformation implementation.
- *Process_input* is the method that creates the input for the PM execution
- *Create_result_model* is the process-to-result transformation implementation.
- *Create_result* is the implementation of the result-to-dashboard transformation.
- *Process_output* processes the output of the PM to a PM model instance.

Each method will be discussed separately. For some complex parts of the code pseudocode is added in separate boxes.

```
def post(self, request, **kwargs):
    data_set_id = kwargs['data_set_id']
    options = json.loads(request.POST.get('selected_options'))
    create_user_specification_model({
        'data_set': data_set_id,
        'options': options
    })
    process_input()

    process_output()

    create_result_model()

    return JsonResponse({
        'result': create_result()
    })
```

Figure 25 POST method which implements the user workflow analysis

6.5.1 Preparation-to-specification

The input for this transformation is the JSON object with the user specification data and the output is an XML model instance of the user specification metamodel (and inherited the instances of the plugin specification metamodel). This method contains all the information about the selected options and the parameters and what this means for the PM execution. All the option and parameter ids are saved as variables to reduce the risk of using the wrong id.

The transformation can extract from the input JSON object which data set is selected. Based on the data set, the transformation knows what options to expect, since this is preprogrammed. The options and parameters used in this transformation are the same (including their ids) as in the user specification interface. For now, these data and ids of the options and parameters are duplicated which brings a large risk. The sharing of these data sets, options, and parameters must be carefully considered in an operational implementation.

When an option is selected in the user preparation interface, this option id is included in the JSON object. The implementation loops through all options in this JSON object and knows that an option is selected when the id is in the object. The implementation knows what code to execute based on this option id. For each option, three steps are executed.

Figure 26 shows an example of this implementation for option 1. The steps are separated with a new line and a comment.

First, the parameter values are extracted from the JSON object. Since the user specification metamodel also recognizes options and parameters in the same construct as the JSON object, these data can directly be used to create the model instances of these options with respect to the user specification metamodel.

```
if ME_OPTION_MOST_USED_ELEMENT in options:
    option = Option(option_id=int(ME_OPTION_MOST_USED_ELEMENT))
    parameter_values = options[ME_OPTION_MOST_USED_ELEMENT]
    param = Parameter_spec(parameter_id=int(ME_OPTION_MOST_ELEMENTS_PARAM_MINIMUM_TIMES), value=parameter_values[0])
    option.parameter.append(param)
    param2 = Parameter_spec(parameter_id=int(ME_OPTION_MOST_ELEMENTS_PARAM_LIMIT_RESULTS), value=parameter_values[1])
    option.parameter.append(param2)

    # THIS IS WHERE THE PROCESS MINING INTELLIGENCE IS ADDED
    param_plugin_1 = Parameter_plugin(name='Attribute name', value='elapsed_time')
    input_plugin_1 = Object_plugin(name='Log data', tool='SUA')
    output_plugin_1 = Object_plugin(name='Transformed log', save_as='elapsed_time.xes', tool='SUA')
    plugin_1 = Plugin(name='Add Time between events (Duration) as Attribute to all Events', tool='ProM')
    plugin_1.parameter.append(param_plugin_1)
    plugin_1.input.append(input_plugin_1)
    plugin_1.output.append(output_plugin_1)

    #THIS IS WHERE THE PLUGIN IS SPECIFIED
    plugin_spec_1 = Plugin_spec(order_number=1)
    plugin_spec_1.plugin = plugin_1
    option.plugin.append(plugin_spec_1)

data_set.options.append(option)
```

Figure 26 Preparation-to-Specification transformation for most used processes and documents

Next, the PM intelligence is added. In the design the plugins can be defined separately from the user specification. In this prototype implementation, the plugin definition happens for each option while processing the specification, but the design does not force this.

Based on the option id, the implementation knows what plugins, inputs, outputs and parameters must be defined according to the plugin specification metamodel. In the example, one plugin is needed: 'Add time between events (Duration) as Attribute to all Events' in the ProM tool. The input is the log data from the SUA and the output is a transformed log. The `save_as` attribute for the output definition tells us how the exported file is called, so we can use it in the other methods. In this implementation, this specification only leads to instructions for the PM execution. Therefore, the names and attributes can be very descriptive. When the PM execution is completely automated this may need more standardization and formal notations.

Finally, the plugin specifications are added for this option in the user specification model. All three options only have one plugin, so the order number is not relevant in this prototype implementation.

All options from the input JSON object are processed in a similar way and added to a data set instance, which itself is added to an instance of the analysis class. The analysis instance is serialized to an XML resource.

For each option, this implementation contains the PM plugin that needs to be used. Which plugin to use is up to the SUA owner. In this prototype, we researched the different plugins available. The plugins which seem to have the most added value, and which had an exportable result (which is often not the case) were chosen.

6.5.2 Process input

Within this method, the step to PM execution is made. Eventually, the user specification model should automatically trigger the correct plugins in the PM tools. However, this is not (yet) implemented and it is therefore a manual intervention for now. To illustrate how it should work and to give the opportunity also for other people than the researcher to execute the PM plugins, the user specification is now translated to textual instructions printed on the terminal.

Figure 27 shows how the instructions look like for the three options of the prototype.

```
PROCESS MINING INSTRUCTIONS
For option with id 1:
  Execute plugin Add Time between events (Duration) as Attribute to all Events from tool ProM with the following parameters:
    Parameter Attribute name with value elapsed_time
    Input: Log data from tool SUA
    Output: Transformed log from tool SUA, save as: elapsed_time.xes
For option with id 2:
  Execute plugin Add Time between events (Duration) as Attribute to all Events from tool ProM with the following parameters:
    Parameter Attribute name with value elapsed_time
    Input: Log data from tool SUA
    Output: Transformed log from tool ProM, save as: elapsed_time.xes
For option with id 3:
  Execute plugin Interactive Data-aware Heuristics Miner (iDHM) from tool ProM with the following parameters:
    Parameter Set dependency with value 1
    Input: Log data from tool SUA
    Output: Causal net export from tool ProM, save as: result_option_sub.cnet
```

Figure 27 PM instructions

For these three options, the parameter values of the user specification only influence the process-to-result transformation, not the execution of the PM plugins. Also, in this prototype a manually collected data set is used and not a continuous stream of new log data. We highly preferred that in this prototype the execution of the UWAT does not have to wait for the PM execution, and since the execution will not change with different settings or over time, we decided to already execute all the PM plugins according to the instructions upfront. The output files were added to the environment and the UWAT could continue without waiting for human intervention. Of course, when the parameters influence the execution, or when an up-to-date data stream is used caching the PM results is not possible.

In all cases, the input is the log data from the SUA. This will in all cases be the input data for this first plugin, since this is the oil for the PM engine. The UWAT assumes these data is already available. Guidelines for the preparation of these data and the implementation for the case study are discussed in Section 7.2.

6.5.3 Process output

This method is a Python script to convert the output files of the PM tool, in this case ProM, to a PM model compliant with the PM metamodel. This method is implemented as a sort of text-to-model transformation. However, creating the PM model is usually not part of the UWAT and is the responsibility of the external call to the PM tools. Therefore, this transformation is not part of the UWAT transformation chain and only implemented for the prototype.

If a file exists with the name of the save_as attribute in the user specification model, the script knows that the option is selected. For this option, the script also knows what information to extract. In this case, if elapsed_time.xes exists the script checks the frequency and duration. If result_option_sub.cnet exists, the script extracts the discovered process for the sub elements. The results of all options are eventually collected in one PM model instance of the PM metamodel.

In the case of option 1 or 2, we want to add a node, representing the place visited relation in the PM metamodel, to the result. For each node, there are two attributes: frequency and duration. These names are the keys, the values are the amount of times that element is visited in the event log and the total duration of users visiting. The duration is saved in the `elapsed_time` attribute in the input event log.

Pseudocode option 1 and 2

```
create new list places_visited
if file exists:
  for trace in traces:
    for event in events:
      duration = event.elapsed_time
      if node exists in places_visited where node.label is event.name:
        attribute_frequency = attribute_frequency + 1
        attribute_duration = attribute_duration + duration
      else:
        create new_node
        new_node.label = event.name
        new_node.attribute_frequency = 1
        new_node.attribute_duration = duration
        add new_node to places_visited
```

In the case of option 3, we want to create a process as defined in the PM metamodel. This is a discovered process, as is described in the PM techniques. For each visited model element, we want to create a single node, also when it is visited more than once. For all transitions, also a transition needs to be added in the resulting PM model. For nodes and transitions no attributes are needed for this option. This discovered process contains the option id, so the process-to-result transformation knows that this is the discovered process of option 3.

Pseudocode option 3

The input file is a causal net file. This causal net contains amongst other nodes, which represent the model elements, and arcs, which represent the transitions.

```
if file exists:
  create new_process
  new_process.option_id = 3
  create dictionary_nodes
  for causal_node in causal_net_nodes:
    create new_node
    new_node.label = causal_node.name
    add new_node to new_process.nodes
    add new_node to dictionary_nodes
  for causal_arc in causal_net_arcs:
    source = find causal_arc.source in dictionary_nodes
    target = find causal_arc.target in dictionary_nodes
    create new_transition
    new_transition.source = source
    new_transition.target = target
    add new_transition to new_process.transitions
```

After both files are checked, the created objects are added to the result and the PM model is ready. The result model can then be serialized for the process-to-result transformation to use.

6.5.4 Process-to-result

This method is the model-to-model process-to-result transformation that processes all these data and adds this to the result model. The result model is only a textual representation.

The implementation contains two helper methods. The helper methods are specifically implemented for the SUA of the case study. In the first helper method, the labels of the PM model nodes are parsed to a nice visual representation for the results. The second helper method identifies whether two nodes are parent and child in the SUA. For now, this is only implemented for direct relations. In both helper methods the input is the label of the PM model instance. These are in the form TYPE:ID where type is the type of the model element and id the id of the model element. This combination is unique. By this combination the content element can be found in the SUA model instance.

Pseudocode helper methods

The content elements are the content elements from the SUA model.

```

helper_1(label):
  for content_element in content_elements:
    if content_element.type is label.type and content_element.id is label.id:
      return label.type + '(' + label.id + '): ' + content_element.name

helper_2(sub_label, parent_label):
  if sub_label.type is parent_label.type:
    for content_element in content_elements:
      if content_element.type is sub_label.type and content_element.id is
sub_label.id:
        if content_element.super_element is parent_label.id:
          return true
  return false

```

This transformation matches the data set of the user specification with the data types of the result model. Based on this match, the scripts know what options to expect. The implementation loops through all options that are part of the user specification.

Pseudocode option 1

The first and second option are implemented the same. The only difference is whether frequency or duration is used.

```

if option_id is option_frequency:
  create new_result_list
  new_result_list.description = 'Most used processes and documents'
  new_result_list.ordered_by = 'Frequency'
  create dictionary_nodes_threshold
  for node in process_mining.places_visited:
    if node.frequency >= param_threshold:
      add node to dictionary_nodes_threshold
  sort dictionary_nodes_threshold on frequency
  limit dictionary_nodes_threshold to param_limit_to
  order_number = 1
  for node in dictionary_nodes_threshold:
    create new_result_element
    new_result_element.name = helper_1(node.label)
    new_result_element.order_number = order_number
    create new_result_element_attribute
    new_result_element_attribute.key = 'frequency'
    new_result_element_attribute.value = node.frequency
    add new_result_element_attribute to new_result_element.attributes
    add new_result_element to new_result_list
    increment order_number with 1

```

For option 1 and 2, the outcome is a result list. They are implemented very similarly. The threshold for the frequency or duration respectively can be retrieved from the parameter of the user specification model. For all places visited in the PM model, the nodes with the respective parameter above the threshold are saved. After all nodes are processed, the list is sorted on the frequency or duration respectively and limited to the amount specified in the user specification. For each node in the limited list a result element is created and added to the result list. This result list is then added to the total result.

For option 3, the outcome is a trace collection. Based on the option id, the correct process in the PM model can be found. The implementation loops through all the transitions in the process. The second helper method checks whether this is a parent-child transition. Then, for every source, the parent, all the targets, the children, are saved. For all different sources, a result trace is created where the source is the parent and the targets are the children, which are added to the trace collection. This trace collection is then added to the total result.

Pseudocode option 3

```
if option_id is option_sub_elements:
    create dictionary_source_targets
    process = process_mining.sub_elements_process
    create new_result_trace_collection
    new_result_trace_collection.description = 'Used subelements'
    for transition in process.transitions:
        if helper_2(transition.target.label, transition.source.label)
            if source in dictionary_source_targets:
                targets = dictionary_source_targets[source]
                if target not in targets:
                    add target to targets
            else:
                targets = new list with target as first element
                add (source, targets) to dictionary_source_targets
    for (source, targets) in dictionary_source_targets:
        create new_result_element_source
        new_result_element_source.name = helper_1(source.label)
        create new_result_trace
        new_result_trace.source = new_result_element_source
        for target in targets:
            create new_result_element_target
            new_result_element_target.name = helper_1(target.label)
            add new_result_element_target to new_result_trace.targets
        add new_result_trace to new_result_trace_collection.traces
```

At the end, the result is serialized to an XML model with the PyEcore library and the result model is ready.

6.5.5 Result-to-dashboard

The final method is the model-to-text result-to-dashboard transformation. The only purpose of this transformation is to translate the result model to a JSON object that could be used for visualization. A JSON dictionary object is created as defined in Section 5.3.4.

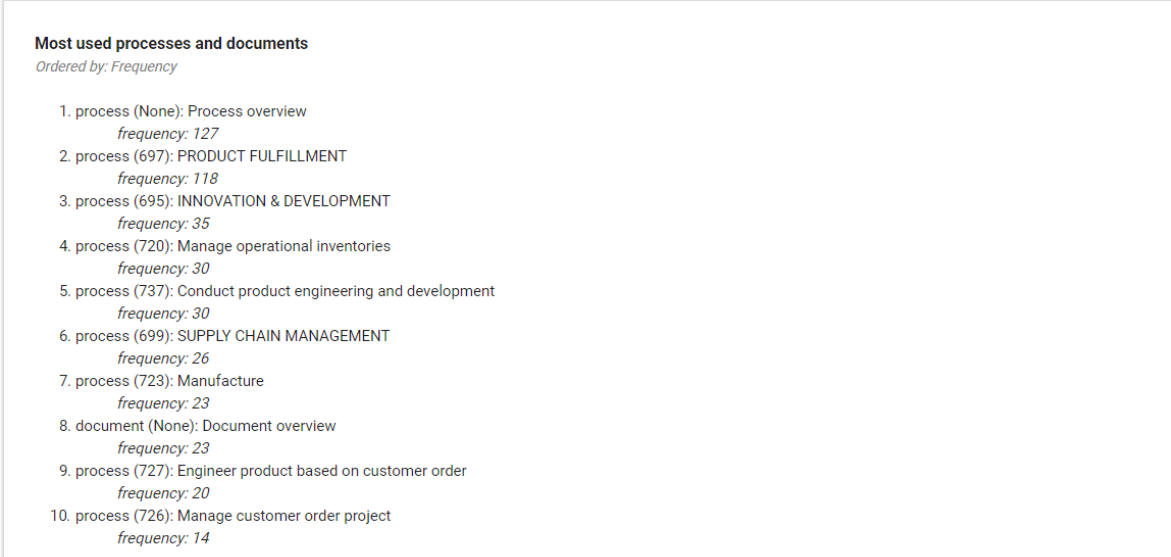
Since this dictionary has the same structure as the result metamodel, this transformation is easy to implement. The implementation first loops through all lists in the result model. For each list, the description, the ordering and the elements, with their information and attributes, are saved. Then the implementation loops through all trace collections. For the trace collections there is a similar approach. For each trace collection, the description and the traces, with their elements and attributes, are saved. All lists and trace collections are added to the result instance.

This implementation returns this result dictionary, which is the outcome of the UWAT. Remember that in this transformation no processing of the data happens; it is only saving the data in a dictionary, so it could be parsed in the result dashboard.

6.6 Result dashboard

In this implementation, the result dashboard is, just as the user preparation interface, part of the Mobina environment. It is also included in the same view as the user preparation. The POST method, called by the user specification interface, returns a JSON object with the results from the result-to-dashboard transformation to the frontend as soon as it is done. This is detected by the AJAX call. Then, the result data is loaded in the VueJS component and for the user workflow analyzer the results for the data set, in this case 'Processes & Documents', shows up.

In the expandable body of this result, first all the lists in the result are printed. Figure 28 shows the result for option 1, the most used processes and documents, in the lay-out used in the prototype. The lay-out is the same for every result list. In this result, you only see one relevant attribute, the frequency. From the metamodel definition this can be unlimited.



Most used processes and documents	
Ordered by: Frequency	
1. process (None): Process overview	frequency: 127
2. process (697): PRODUCT FULFILLMENT	frequency: 118
3. process (695): INNOVATION & DEVELOPMENT	frequency: 35
4. process (720): Manage operational inventories	frequency: 30
5. process (737): Conduct product engineering and development	frequency: 30
6. process (699): SUPPLY CHAIN MANAGEMENT	frequency: 26
7. process (723): Manufacture	frequency: 23
8. document (None): Document overview	frequency: 23
9. process (727): Engineer product based on customer order	frequency: 20
10. process (726): Manage customer order project	frequency: 14

Figure 28 Result dashboard list

The trace collections are underneath. All lists and trace collections are separated using a horizontal line. Figure 29 shows an example of how the trace collection looks like in the result dashboard, in this case the results of the option used subelements. For the trace collections the description is printed as the title. Next, all the sources are shown as an unordered list, since the trace collection does not recognize order. Below each source, all the targets can be found. In this example, these are the used sub elements.

Used subelements

- process (727): Engineer product based on customer order
process (1115): Manage engineering tasks and deliverables
- process (737): Conduct product engineering and development
process (965): Develop detailed design
- process (695): INNOVATION & DEVELOPMENT
process (737): Conduct product engineering and development
- process (697): PRODUCT FULFILLMENT
process (727): Engineer product based on customer order
process (723): Manufacture
process (720): Manage operational inventories
- process (720): Manage operational inventories
process (1216): Receive and unload incoming goods
process (1213): Receive and process orders to be picked and issued to production
- process (723): Manufacture
process (922): Execute assembly
- process (None): Process overview
process (697): PRODUCT FULFILLMENT
process (699): SUPPLY CHAIN MANAGEMENT
process (695): INNOVATION & DEVELOPMENT
process (None): Process overview
- process (699): SUPPLY CHAIN MANAGEMENT
process (704): Plan, detail and monitor the supply chain network
- document (None): Document overview
document (707): Company's warehousing systems
- process (704): Plan, detail and monitor the supply chain network
process (1279): Determine supply chain strategy, requirements and targets

Figure 29 Result dashboard trace collection

In this result dashboard, all text comes directly from the result model. In this way, this result dashboard can be used for all different kinds of data or SUAs without having to create a tailormade view for each analysis. Each type of result in the result metamodel has a visual representation. Therefore, if the result metamodel is extended with another type, one only has to define the visual representation once and it can be reused for all analyses of that type.

In this case all the results are pasted below each other. This is possible since there are only three options and because the data sets are small. However, if this keeps growing, this way of representation may not work anymore. Then, there will be a need for filtering, clustering and/or hiding. Since the result dashboard is only part of the implementation to be able to validate the prototype, no attention is given to this aspect. However, for the SUA owner the representation is an important attention point when implementing a UWAT.

7 SUA guidelines

In previous chapters we discussed what SUA owners need to implement in the UWAT architecture specifically for their SUA. But before the UWAT can be used, the SUA owners need to prepare several other things. These are discussed in this chapter. This chapter first discusses the business objectives of the SUA which eventually lead to the options for the user workflow analyzer. Next, it discusses how the software of the SUA should be logged. Finally, it discusses how an artificial data set can be collected in case of testing or when no real data is available. For all three preparation tasks, general guidelines are presented for SUA owners. This is illustrated by how this is done in the case study.

7.1 Business objectives

The SUA owner must determine which analyses to implement in the UWAT. This has consequences for which data to log, which PM plugins to use and most importantly which options the user workflow analyzers have. Before implementing these options, the SUA owner should know what to achieve using the UWAT. For this purpose, the business objectives are introduced. The business objectives represent the questions of content and software developers of the SUA that could be (partially) answered with user workflow analysis.

7.1.1 Determining the business objectives

To determine the business objectives of the SUA, the SUA owner needs to know which questions in the company can be answered by user workflow analysis. These questions should be agreed upon with a wider range of people. We recommend discussing this with a subset of user workflow analyzers who have the conceptual level to understand what user workflow analysis consists of and which questions it can answer.

The UWAT target group are the content and software developers of the SUA. Both groups can have different questions that need to be answered, but questions can also be relevant for both groups. For scheduling reasons, it is sometimes hard to have an integrated discussion of both teams. However, we recommend having at least one person involved who has a cross-disciplinary role and can oversee the impact on both.

One should be careful that the discussion is not about details or the options of the UWAT. The discussion should be about higher-level questions that need to be answered, which can then be translated to options in the implementation phase. When discussing the options, the scope will soon be too limited and the higher goal of the UWAT, improving the content and the software of the SUA, is overseen.

The amount of (sub)questions should not be limited. However, this does not mean that everything should be implemented, in the short term or ever. A broad scope leaves space for questions relevant for the long term. When using the UWAT in practice, the business objectives should probably also be revisited regularly. However, using the UWAT in practice is out of scope for this research.

7.1.2 Case study

For the case study, a list of 14 main questions has been defined, most of them including sub questions. The complete list can be found in Appendix A. Questions for content developers, for software developers, and for the combination have been separated. The sub questions sometimes already hint to which analysis to implement. This shows the risk of already defining options in the discussion. This is not necessarily a problem, but one should be aware of this.

In the case study, the business objectives were discussed with the people who were actively involved in the content or software development. People who only have a reviewing role were not included. The discussion was held for the content and software developers separately. Since the researcher, who was also the SUA owner for the case study, is familiar with both sides, she played a cross-disciplinary role.

7.2 Logging of the software

Input for the UWAT, more specifically the PM, is the event log data. To generate these event logs, the different actions the user takes in the SUA must be logged. Many user actions on the user interface do not lead to separate calls to the server. To get good insight into the usage, it is then important to log the actions at the client-side.

7.2.1 Logging data

Part of the data that needs to be logged is generic and predefined, while another part is specific for each SUA. Which data need to be logged specifically for the application, depends on the business objectives. The necessary information about the content and software components to answer the questions posed should be logged. The standard data that need to be logged is:

- Timestamp, preferably including milliseconds.
- Page/view where the action happens.
- Type of action (e.g., a click on a button).
- Identifier to separate the views, when the user is working in different views at the same time (e.g., different tabs in the case of a web application).

Besides that, other fields are applicable to all potential SUAs and can be interesting for user workflow analysis:

- Anonymous identifier or additional demographic information of the user.
- Session number.
- Type of application (e.g., browser).
- Version numbers of content and software.

A good practice is to log all the loads of a page/view and all clicks that happen in the application. This not only means button clicks, but, for example, also the selection of a text field. One can best log all possible relevant information of the loads and clicks. It's better to have too much information and filter it out later if it is not useful, than missing parts at the end. This also gives the possibility to adapt to future business objectives.

There are two important requirements for the logs of the software: the log should be complete, and the log should contain all information to answer the business questions. There are two useful guidelines to achieve this. Firstly, let another developer or someone in the business check if everything is indeed logged and no relevant information is missed. Secondly, use an iterative process by starting with small data sets. Since quite some data needs to be collected to get useful insights, restart logging every time is not possible. By having test runs with small data sets, possibly created for this purpose only, one can see if the right data has been collected and completeness in production is ensured.

7.2.2 Logging framework

Depending on the nature of the application, an appropriate logging framework must be selected or created. This framework should fit the technology choices of the SUA. One can use an existing logging framework, build a new logging framework or use a hybrid form. Both in business [38], [39] as well as in science [40]–[43] there is a lot of discussion about when to use an off-the-shelf product or to build custom software.

Off-the-shelf products have the advantage that they often have nice features and that these products carry the risk. However, a custom-built framework gives flexibility, scalability, ensures that it has all needed features and there is no overhead of learning the details and structure of the existing framework. Therefore, the biggest consideration is whether off-the-shelf products fulfill all the requirements for the logging framework.

The data logging framework has the following requirements, based also on the data that needs to be logged as described above:

Functional requirements

1. The logging framework should be able to collect a timestamp
2. The logging framework should be able to store the page/view where the action has happened
3. The logging framework should be able to store the type of action that is logged
4. The logging framework should be able to extract and store an anonymous identifier of the user to analyze behavior over sessions
5. The logging framework should be able to extract and store the session number
6. The logging framework should be able to handle the user working in different windows at the same time
7. The logging framework should be able to extract and store the type of application that is used by the user
8. The logging framework should have the possibility to add fields specifically for each application
9. The logging framework should be able to handle freely defined data fields
10. The logging framework should be able to handle different software and content versions
11. The logging framework should have the possibility to extract the log data

Non-functional requirements

12. The logging framework should work with the programming language and environment that is used for the content-intensive application
13. The logging framework should have very little data loss since this gives wrong traces
14. The logging framework should be able to handle the amount of data needed for the content-intensive application
15. The logging framework should make it easy to collect all necessary data
16. The logging framework should collect and store the data in a standardized way to make it usable for analysis

An off-the-shelf product that fulfills these requirements would be recommended. If there is no product that fulfills all requirements sufficiently, letting the in-house software development team build such a logging framework is the best option.

7.2.3 Case study

For Mobina, a JavaScript logging framework was implemented. Some research was done on existing JavaScript frameworks, both commercially and open source. However, since this case study was used as a validation for the architecture, the main focus is that there is no compromise on what can be logged with the software, as well as how it should be stored and exported. Besides that, the requirements could also change in time, so there is a large need for flexibility. No existing framework fulfilling all requirements was found, and therefore a custom logging framework was built directly in the Mobina application.

The actions were logged in JavaScript with support of jQuery and the backend was built in Django⁷ and Python. The Django model specified all the data that should be logged. The database could easily be exported to a JSON object. To support standardization, a standard JavaScript function was defined which was used to log all relevant actions and which ensured all the necessary data was logged. All data that can be collected automatically and which contains no specifics for the actions logged, e.g., the datetime, the browser type, etc., is added to the request and does not need to be implemented specifically for each log action. Every 30 seconds or when a session is started, the log is sent with an API call to the centralized Django server. Then, the Django server adds all the standard information and stores the log actions in the database. By using this short time frame, in combination with Mobina's scalable framework, all non-functional requirements are satisfied.

All functional requirements are satisfied, but this depends mainly on the definition of the data model. Figure 30 shows the data model as defined in Django. This is a combination of generic fields as defined in Section 7.2.1 and Mobina-specific fields. This model does not contain all generic fields, e.g., software and content versions are missing. The content and software versions were not defined yet on the moment of logging, so therefore they are not yet included in the implementation. This does not limit the research, since it is only a proof of concept.

```
class Log(models.Model):
    datetime = models.DateTimeField(auto_now_add=True)
    server = models.CharField(max_length=50)
    page = models.CharField(max_length=100)
    type = models.CharField(max_length=32)
    element = models.CharField(max_length=32)
    identifier = models.CharField(max_length=32)
    data = models.CharField(max_length=1000)
    user_hash = models.CharField(max_length=128)
    session = models.IntegerField()
    pageload_identifier = models.CharField(max_length=32)
    browser = models.CharField(max_length=32)
    os = models.CharField(max_length=32)
    is_mobile = models.BooleanField()
    is_tablet = models.BooleanField()
    is_pc = models.BooleanField()
```

Figure 30 Data model logging framework

For the understanding of the case study, some fields are discussed shortly. The field *server* shows which Mobina server is used (e.g., production or development server). *Type* and *element* represent the type and id of the content element, e.g., a process. This combination is always unique. This is always relevant and thus stored for every log item. *User_hash* is the hash for every user. This can in no way be tracked back to any user but gives the opportunity to track a user's actions throughout multiple sessions. *Pageload_identifier* gives the possibility to handle a user working in multiple tabs. Additionally, some technical details are saved like operating system, type of device, and type of browser. This is both to differentiate between behavior, but also to be able to filter out data if something, for example, only works in one browser. Finally, the *data* field stores information in a JSON format that is specific for the action that is logged. This can be different per action or item and is therefore free-format.

⁷ <https://www.djangoproject.com/>

In this implementation all page loads and all clicks in the web application are logged. Besides this, the occurrence of the focus and blur events are also registered so one can see when they start and stop working on, for example, a comment. For all relevant pages, all possibilities on that page are logged, even though not all information is necessary to answer the business objectives. The logging can be tested through a specific page in development mode.

Figure 31 shows an example of the logging in JavaScript. In this case, a click on the process in the reference model is logged. The method *Mobina.jslog.log* is the standard method for logging in JavaScript. The first attribute shows the type, i.e., this is a click, the second attribute is the action, i.e., a click on a process, the third attribute is the id of the element where the click happens, and the fourth attribute is the freely defined data. These data is thus specific for the click on a process but can be used in the rest of the analysis.

```
//Click on process
b.on('click', '.process-card', function () {
  Mobina.jslog.log(CLICK, 'click_process', $(this).closest('.page-card').attr('data-id'), JSON.stringify({
    "app": COLLABORATION_APP,
    "event": "click_process",
    "type": $(this).closest('.page-card').attr('data-type'),
    "type-element": PROCESS,
    "id-element": $(this).attr('data-id'),
    "sub": false
  }));
});
```

Figure 31 Example JavaScript log

7.3 Data generation

Sometimes no real data is yet available. There might be a company-specific reason like waiting for a new release, or the UWAT needs testing before moving into production. In that case, a data set needs to be artificially collected. This has been done in our case study. The purpose is to create a representative data set, while keeping the effort to create the data sets as low as possible. This section assumes the data set is for testing the quality of the analysis. When it comes to testing, for example, the performance of the UWAT, different experimentation variables should be used.

The data generation is discussed based on the experiment process as proposed by Wohlin et al. [44], which is shown in Figure 32. The experiment is the data generation. When using this for the UWAT architecture, the analysis and interpretation step is done implicitly by using it in the architecture. Therefore, this is not discussed anymore in this section.

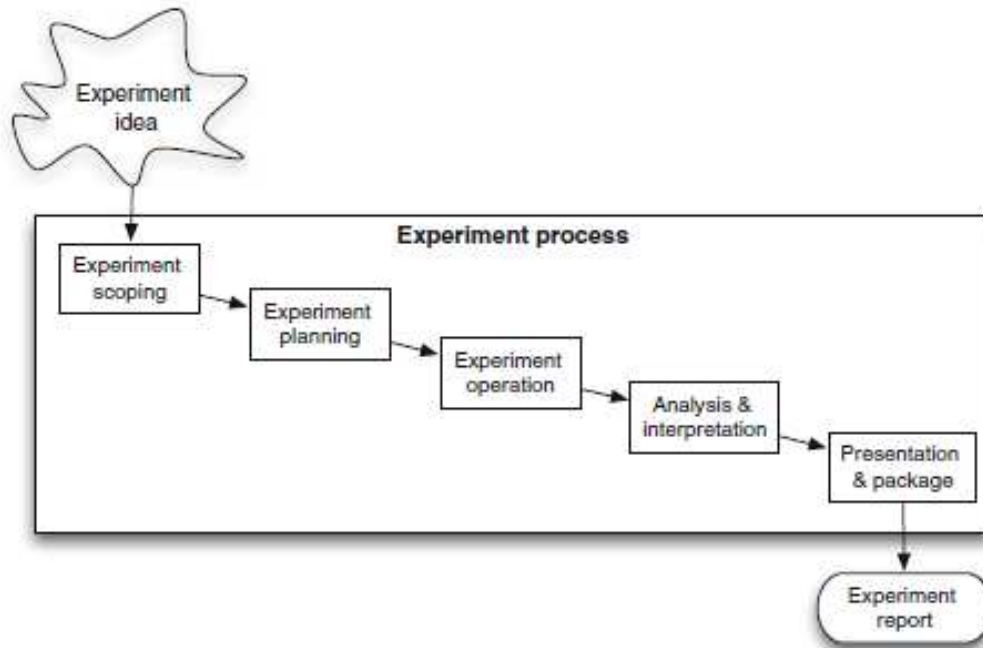


Figure 32 Experiment process

7.3.1 Experiment scoping

The experiment scoping has as output the goal definition. This consists of five parts [44]:

1. *Object of study*. In this case the SUA.
2. *Purpose*. To collect a representative event data set as input for the UWAT.
3. *Quality focus*. The data set needs to fulfill five requirements to ensure the quality:
 - a. The data set needs to be representative
 - b. The set must be large enough to draw actual conclusions
 - c. The data set must also contain edge cases to ensure all behavior is included
 - d. The number of steps per session should be sufficient
 - e. A session is from the moment a user logged in until the moment the user logs out or otherwise quits working on the SUA (e.g., closing the application)
4. *Perspective*. The perspective is the user workflow analyzer
5. *Context*. The context is the SUA that is used by actual users of the application.

7.3.2 Experiment planning

The experiment planning has as output the experiment design. It consists of several steps [44], but since this data generation is a smaller research than is often used for this framework and since this experiment is not empirical, only the four relevant steps are selected here.

First is *context selection*. Since it is important that the data set is representable, the data collection should happen by people that are actual SUA users or are representative for SUA users. Besides that, also the setting should be as if it was actual use. This means that own devices should be used, the SUA is used as it is intended (i.e., through a web browser if it is a web application), no extra support from the person who collects the data is given, and it needs to be done in the environment where the SUA would be used (i.e., in their own office).

Secondly, the *variable selection*. Next to requirements for the space as discussed with context selection, the most important variable is the amount of data to be collected. As described in requirement 2 and 4 the amount of data must be large enough to draw actual conclusions. The amount of data depends on the number of sessions and the number of steps taken per session. Because these factors depend heavily on the nature of the SUA, it

is recommended to look at the total time of data that needs to be collected. On the one hand a large data set is preferred, on the other hand the investment for collecting the data set should be low. For this kind of data collection, a recommendation would be to collect between 5 and 10 hours of data.

Third is the *selection of subjects*. As mentioned before, the participants should reflect actual users. Most preferable are actual users, but another possibility is to use people with the insights of real users and/or who can think like an actual SUA user. Important is that the subjects have a feeling with the content of the SUA. To ensure that the data also contains different traces and edge cases, it is also recommended to use people with different backgrounds that represent the whole population of end users.

Finally, the *experiment* must be *designed*. To ensure the data contains enough differences, but at the same time enough similarities to draw conclusions, some participants should perform the same assignments. In this way, there is a high probability that there are some similarities. However, the assignments should be freely defined, so that participants can use their own approach to create differentiation. A recommendation is to use scenarios, where the participants are a type of user and are nudged into the right direction. Important is that the scenario is representative for what the SUA users would use the application for. Every assignment should be a session. Also, the assignments do not need to be of the same length, since this will also differ in real use.

Important in the experiment is that the participants should clearly understand upfront what the data generation is for and what is expected from them. A part of this explanation is also that there are no wrong ways and that different or unexpected behavior is part of the assignments and the data collection. Finally, the participants should understand the SUA and know what the scope of the experiment is, e.g., when only one part of the SUA should be used, this should be communicated upfront.

7.3.3 Experiment operation

Experiment operation consists of three parts: preparation, execution, and data validation [44]. However, since the data has as purpose to test the application, data validation is not done and discussed as part of the experiment but is part of the bigger research.

In this case, the *preparation* consists of three parts: the technical preparation, the environment preparation, and the preparation of the participants. The technical preparation is about making sure the software is logging properly. On the one hand, the environment preparation is about creating the technical environment and accounts for the participants. To make sure the behavior is not influenced and leads to too many similar traces, we recommend having a separate environment for each participant. If different types of users are used for the assignments and one wants to find some relations to that, separate accounts should be made for each participant for each assignment. On the other hand, environment preparation is about putting default data in the application. For example, if someone should collaborate with another fictional user, some data of this user may also be interesting to store in the database. The need for this depends on the business objectives of the analysis and the nature of the application.

Finally, the preparation of the participants. Before the participants start the assignments, they should have the relevant information. This consists of three parts:

- Description of the data collection including the goal, what data is collected and what the collected data is used for.
- Relevant information about the application. This may contain some sort of manual, links to support or the scope of the experiment.

- The assignments. The assignments should be described for the participant. If information is needed about the environment, the type of user or data that is already in the server, this should be included in the description of the assignments. A rough time estimation for the assignments in total or per assignment can help ensure the participant he/she is doing the right thing. However, emphasize that if they believe shorter or longer is more appropriate, this should be done since this reflects actual behavior.

The execution phase starts by presenting all the information to the end user, if this was not done yet, and assuring there are no questions left. After everything is clear, the credentials to log into the SUA can be delivered. From this point on, the data collector should only be there for technical support. Questions about the assignment or the application should not be answered, so the participant is not influenced. The rest of the data collection depends on the user.

7.3.4 Presentation & package

The presentation & package section is part of the overall research. However, it is important to mention here that the data should be delivered in a format that can be used by the rest of the architecture. Otherwise, an extra conversion script needs to be developed to prepare the data in the correct format.

7.3.5 Case study

Since the Mobina application was not yet used by customers when the research was in the generation phase, the decision was made to collect data artificially. Five scenarios were developed in which each participant was asked to play the role of a certain person in a fictional company that uses Mobina for a certain purpose. All scenarios were for the same company, a crane company, but all had different perspectives. The assignments can be found in the manual for the participant, which can be found in Appendix B.

For the data collection, five participants are selected. Since some of the content is confidential and since some basic knowledge of the content of Mobina was required, the people selected were all from the Mobina team. Without the basic knowledge, the startup curve would be too high, which could lead to unreliable data. Because of the complex content and software, the decision was made to use fewer people who could do multiple scenarios. Each participant got the exact same five assignments, which were estimated to take about 1.5 hour in total, leading to approximately 7.5 hours of data collection. No time division was made between the scenarios, because this will also differ in real use. The five participants were selected to reflect different backgrounds and characteristics, which was expected to give differences in the collected data even though they got the same assignments. The five participants can be found in Table 1. The five participants had several differentiating aspects: age, role in Mobina, experience with the target group, and background next to Mobina.

Table 1 Participants data collection

Jasper Boot BSc	Jasper Boot is 22 years old and is software developer at Mobina. Next to his work at Mobina he is a Master student Computer Science at the University of Twente.
Ir. René Hol	René Hol is 58 and is co-founder and CEO at Mobina. He has a lot of experience in management consultancy for industrial companies, both in business and IT.
Valerija Olsevaska MSc	Valerija Olsevaska is 24 years old and is knowledge modeler at Mobina. She graduated in the summer of 2017 for the master Technology and Operations Management at the University of Groningen.

Jochem Verburg MSc	Jochem Verburg is 25 years old and is co-founder and advisor of Mobina. He finished his master Business Information Technology at the University of Twente in February 2018 and is currently working as product manager at Voortman Steel Machinery.
Prof. dr. ir. Hans Wortmann	Hans Wortmann is 67 years old and is knowledge contributor at Mobina. Besides his work at Mobina, he is chair professor Information Management at the University of Groningen. He has years of experience in both practice and science in the manufacturing industry.

A meeting of 2 hours was scheduled with all participants. They did not receive any information before the meeting. At the start of the meeting they received a document (see Appendix B) which gave them more information about the research, the application and the assignments. The information about the research was also explained verbally. The information about the software was more background information for the participants who were not yet that familiar with the software. This also presented the scope of the research. Next, the scenarios and the fictional company were introduced, as well as a list of other people that worked in the company and the information systems that were used in the company (all fictional data). This information could be used in the scenarios.

Nothing extra was explained and no questions were answered about the scenarios and what they should do with the software. It was strongly emphasized at the beginning of the assignments that they should imagine they were the people in the scenarios and should use the software as they thought the person in the scenario would. It was also emphasized that different behavior did not matter.

For this data generation, a separate server was set up. All participants could access the server through the Google Chrome browser on their own laptop. Before the meeting a separate demo company was created for all the participants, as well as different login credentials for each scenario and each participant. The credentials were available on paper at each meeting. At the beginning of each assignment they were asked to login with the credentials for that scenario, and then log out after they were finished with the assignment. To also collect some collaboration data, some demo data was entered before the meeting at pages the participants were expected to visit. The participants could interact with these data of other fictional employees of the demo company.

At the end, all the data was collected successfully and stored in the database of the logging server. After each meeting, a back-up was made by exporting the data as a JSON file. After all meetings, the final JSON file could be exported.

8 Validation

This chapter discusses the validation of the design presented in this thesis. Firstly, the two approaches used for validation are introduced: user validation and technical review. Next, the validation is discussed per requirement, as introduced in Section 4.1. Finally, the validation results are summarized.

8.1 Approach

In this research the design of the UWAT is presented, and an initial prototype is implemented. This is an early stage in the design cycle presented by Wieringa [45]. Wieringa presented different validation methods for design studies [45].

The added value of this design can only be achieved when its users are able to grasp the added value of the UWAT, so it should be presented to potential user workflow analyzers. In this phase in the design cycle, it is important to analyze the potential of the design without needing to implement a perfect, fully operational application. Therefore, it's useful to present it to users who can look past its potential flaws. Based on the presented validation methods, we believe expert opinion is the most appropriate option. We use a representative group of users as experts in a user validation to assess the design and prototype.

Additionally, some requirements are about the potential the design offers for a good implementation. Since only a prototype is implemented, this will not in itself be representative to validate all requirements. The prototype might also fulfill certain requirements which does not necessarily mean the design supports these requirements. Therefore, a careful technical review of the design, including the used techniques, and its impact on implementations is needed to draw conclusions. This technical review is also done through expert opinion.

8.1.1 User validation

In the user validation, the design of the artifact is the prototype that was developed. The experts of the user validation, called *user experts* in the rest of the validation, are the potential user workflow analyzers. In this case, the user experts are the content developers of Mobina.

The first research question is to discover how user workflow analysis can help content and software developers of content-intensive applications to gain additional insights in the application and its content. The added-value of this type of analysis can only be determined by people who know the relevant parts of the application very well, in the case of the prototype the reference model. This makes the role of experts in the validation crucial, since they are the ones that can give insights in the added value.

Besides that, the combination of PM and MDE are expected to allow the creation of a user-friendly solution. User-friendliness does not relate to the interface design (since this is not part of the UWAT) but is about whether it can be used without knowledge about the implementation or the techniques used and whether this way of analyzing and specifying the analysis gives the user workflow analyzers the feeling they can get the expected and desired results. To objectively evaluate this, the perspective of an actual user workflow analyzer is needed. User validation can provide these insights for the validation.

The three experts involved in this validation are René Hol, Jochem Verburg and Valerija Olsevskaja (see Section 7.3.5). They were involved in the determination of the business objectives for Mobina. They are potential user workflow analyzers since they co-develop the content of Mobina. There were no other potential experts available in Mobina. To assess the added value of the results, in-depth knowledge about the content is necessary and they are the only people with enough knowledge about this.

All three user experts are interviewed separately in a semi-structured interview. In a semi-structured interview there are no strict questions, but instead there is an open discussion within a framework. At the beginning, they first get a recap of the research and what we want to achieve. The business objectives within scope are introduced as well as the analyses implemented to achieve these business objectives. The user experts should now know what to expect. Then we walk through the prototype one step at a time. Since only a few options are implemented, these are predefined, and the user experts have a limited choice. Notes are made during this walk-through about relevant statements of the user experts. We then discussed the prototype, where the relevant topics have been brought up. Thanks to the nature of a semi-structured interview, a lot of different opinions could be gathered this way.

8.1.2 Technical review

The second part of the validation is done using a technical review. The expert in the technical review is the researcher of this report, since a complete view on the design choices and what impact these choices have is necessary to evaluate the design and its implementation.

The technical review is a critical review of this design, and where necessary its implementation, to validate to what extent the requirements of the architecture are fulfilled. The technical review is also an important part of the validation, since this validates the combination of PM and MDE and the desired effects of their combination on aspects like reusability, scalability and user-friendliness.

8.2 Validation results

The validation is discussed per requirement. The different types of validation are used for the following requirements (can be both):

Validation type	Requirements (see Section 4.1)
User validation	2, 5, 6, 7, 13, 14
Technical review	1, 3, 4, 5 -12, 15, 16

8.2.1 Functional requirements

1. *The system should be able to extract the specified patterns from the input data of the system.*

The MDE metamodels and proposed transformations are capable of processing and modelling all kinds of complex relations due to the generic setup of all metamodels. All possible patterns are theoretically processable by the UWAT, but practically it does depend on the implementation by the SUA owner.

An important limitation is the external PM plugins. Not every possible relation can be extracted yet by these PM plugins, and not everything can be exported and transformed to a PM model yet. Therefore, maybe not all desired analyses can be implemented by this architecture. However, PM is quickly evolving in both implementation and research, so this limitation is expected to become less of a problem in the future.

2. *The system should be able to give the end users insights in the data that cannot be directly obtained from the data.*

All three user experts had a similar conclusion: the analyses presented here are still very simple, but they already show an added value; i.e., it can help them improve the content of Mobina. An example of the added value was that the results of the analysis 'most time spent at processes and documents' showed that a lot of time was spent on the process and document overview. For the process overview this was expected, since this is the opening screen. However, with the document overview this was surprising. A possible explanation here was that the document overview was maybe not clear. The results of

the analysis 'Subelements used' showed that only some of the subprocesses were used. A potential conclusion for this was that there was maybe too much detail in the subprocesses already. More analyses and data are necessary to confirm this, but for the user experts this already gave interesting additional insights.

All user experts mentioned that adding more advanced relationship analyses could provide even more added value. In the prototype, only direct relations to the subelements were explored. The user experts however mentioned that it would be interesting to see if there is a connection between different parts of the reference model. All three also mentioned that the added value depends on what options are available and how they are implemented.

Some of the basic analysis, like how often a process is visited, can also be requested in different manners. However, when looking at the relations between different elements, like at the third option, no one could think of a proper alternative way how to achieve this.

There were also several remarks on the results. Currently, the analyses are presented separately. However, the user experts mentioned that the crosslinks, i.e., combining the results, are more interesting. If this technique can make this possible, then this adds more value. One of them mentioned that the most added value is that own expectations about the behavior can be compared with the actual behavior which is shown in the results. The results trigger this comparison. Finally, interactive behavior with the results adds more value. It could lead to more insights if one can filter the results, compare them over time, etc.

3. *The system should be able to process the data in a deterministic way.*

Whether this is guaranteed depends on two things: whether the PM plugins used are deterministic and whether the implementation of the UWAT is deterministic.

The PM plugins are not implemented by the UWAT implementer and their implementation is not influenced by the input of the UWAT. It is therefore crucial when selecting the PM plugins for the different options to check whether the plugin satisfies all requirements and thus also whether the plugin handles the data deterministically.

As for the rest of the design, the UWAT is implemented deterministically for the part that is standardized. The metamodels enable deterministic behavior and the transformations are intended to handle these models in a deterministic way. However, the implementation of the transformations could still lead to nondeterministic behavior.

4. *The system should be able to handle different versions of the input data.*

Versioning is about filtering the data and matching versions; something that can be added on top of the current design. It has no principal influence on the research and design and is therefore left out of scope. A small step towards versioning is already made when adding the version of the data to the SUA metamodel. To implement versioning completely, the version of the log data must be linked with the version of the SUA model instance and handled by the transformations.

5. *The system should be able to provide the results specified by the users.*

The specification by the user workflow analyzers of the required analyses and parameter values is passed to the UWAT via a JSON object. Via a transformation, this can be stored in a user specification model instance. Thanks to the central role of the user specification (meta)model in the different transformations and the PM execution, the user specification is embedded in all parts of the implementation and the specified results are eventually returned.

More extensively, the system is only doing the analyses that are specified by the users and nothing else. The design limits the PM execution only to the plugins that are necessary for the selected options. In the transformations, only if an option is selected, the related code is executed.

With the user experts, the role and the flexibility of the specification was discussed. All three agreed that restricted options were necessary to draw useful conclusions about the data. User workflow analysis is too complex to give the user workflow analyzers more freedom to define analysis (e.g., for example selecting the PM plugin). One mentioned that the conceptual level of the user workflow analyzers probably determines the level of flexibility that is desired by them.

Again, the user experts emphasized that the satisfaction depends on which options and parameters can be used. Basic parameters like the limits and threshold are always useful, but the parameter which gave to possibility to influence what relations in the reference model are considered, gave more in-depth control over the analysis which increased the satisfaction for one of the three user experts.

6. *The system should be able to use the information about the content-intensive application in its analysis.*

The SUA is embedded in this design through the SUA (meta)model. This model is created specifically for each SUA. The entries can then be used in the process-to-result transformation to use the information in the result dashboard. The process-to-result transformation needs to be specifically implemented for each SUA. However, we found a way to standardize part of the transformation. The application model is only used in the helper method. Therefore, only the helper methods need to be implemented separately for each SUA, not the parts that process the PM outcome.

With the user experts, we discussed whether they feel like there is enough SUA specifics in the analysis and the outcomes. They all immediately confirmed that they did not feel like (part of) the analyses were standardized. In the results, it is also clear that it is about Mobina. They added that parameters like those for option 3 strengthens this feeling, since this is clearly specific for the content structure of Mobina.

7. *The system should return a data set with the outcomes of the analyses.*

At the end of the UWAT transformation chain, the result model is translated to a JSON object in the result-to-dashboard transformation. This JSON object contains the results of the selected options by the user workflow analyzer. As mentioned in the previous requirements, this returned data set only contains the results of the selected options. In the result model, the result per option is saved, so a clear distinction is made between the results of the different options.

With the user experts, it is discussed whether the returned data is what they would expect and want. They all expected the results as presented. There were some ideas about the visualization, but this is out of scope for this project. Even though they liked that they could see the results per option, they would also prefer to see the results of the combined options.

8.2.2 Non-functional requirements

8. *The system should need no user interaction after the specification of the analysis.*

The design of the UWAT is set up in a way that no user interaction is needed. The calls to external tools are handled by the UWAT and all the information the UWAT needs comes from the initial user specification.

In the implementation of the prototype however, no automatic calls to the PM plugins were possible yet. Therefore, someone needed to manually execute the plugins. This is a limitation of the implementation, not the design. This was done by the researcher and not the user workflow analyzer, as for whom this requirement was intended.

To exclude all manual intervention, the UWAT must be able to call the PM plugins directly and the results should be processed to a PM model instance. This could be handled by the PM tools or by some sort of middleware which handles all PM specifics. Such an API or middleware was not available at the time of research. The UWAT is designed in a way that PM is executed is an external tool and not part of the UWAT implementation itself. Therefore, the design fulfills the requirement. However, if one wants to automate it completely at this moment, one should develop this himself. We would recommend developing some sort of middleware since then you are independent of a specific PM tool and you can immediately develop the correct PM model instance without needing to adapt the PM tools or plugins.

9. *The system should have as much as possible generic analyses that are applicable to all content-intensive applications.*

The aim of the design was to limit the role of the SUA in the UWAT. An important step is that the SUA specifics are only limited to the SUA (meta)model in the UWAT. The other metamodels include no elements that are specific for Mobina, which makes them applicable to all different SUAs.

The complete standardization is however limited by the definition of the options and parameters. Even though the (meta)models are generic, the (implemented) options are specific for each SUA. Especially the process-to-result transformation is SUA specific, since the SUA details and the options with their PM execution should be known. The preparation-to-specification transformation can be standardized partly because the plugins can be defined separately from the user specification and because the input JSON object has a standard format. However, the translation of the different options to the PM execution depends on the defined options, which are SUA specific. The result-to-dashboard transformation is completely standardized, since the result model is set up in a generic way, as well as the output JSON object. To conclude, still a significant part of the UWAT is not standardized.

10. *The system should be flexible.*

10.1. *The system should be easily extendable with new analyses.*

Due to the flexible definition of options and parameters in the user specification and transformations in the design, new analyses can easily be added from a technical perspective. The complexity of adding new analyses is the conceptual discussion on what to achieve and what this means for the PM. In this design, this is the responsibility of the SUA owner. This is however not limited by the design and implementation of the UWAT, which is what the requirement focuses on.

10.2. *The system should be easily made applicable to new functionality of the product that is analyzed.*

Due to the separation of the SUA specifics from the rest of the UWAT in the SUA (meta)model, new functionality could be added well. This needs to be defined in the SUA metamodel and the SUA model needs to be updated. However, this does not mean that there are already analyses available. If new functionality should be used in the analyses, the transformation must be updated, or new analyses must be defined. This is related to the previous sub requirement.

11. *The system needs to be presented to and used by the user as one system even if it is an implementation containing multiple tools and techniques.*

In this design, the user is not exposed to the implementation and the different tools used, if the preparation interface is implemented correctly and automatically passes the data to the UWAT. Due to the design of the UWAT and the metamodels presented, the design can do the calls to external systems, translate the options and input to what external information is needed, and process this information to useful results. In other words, the UWAT can handle the different systems involved and no manual intervention from the user workflow analyzer is needed here.

12. *The data integrity should remain when data needs to be converted to another format within the system.*

Several data conversions are done within the UWAT implementation. First, the translation of the input JSON object to the user specification model. Here, the user specification data can be compromised. Next, the PM tools start working with the log data and give output. This output data is again translated to the PM model, which can be seen as the translation of the log data to the UWAT. This is then all processed to the result model and finally, this is transformed to the output JSON object.

Whether the data is compromised depends heavily on the implementation of all the different translations. There are two large risks. First, the SUA owner implements some (parts) of the transformations itself, which can lead to the data integrity being violated. Second, an external call is done to the PM tools on which the UWAT has no control. If this part is implemented wrongly, this can also lead to a violation of the data integrity. To satisfy this requirement, it is therefore crucial when implementing the UWAT that this is always tested carefully. Only when all parts of the implementation ensure data integrity, this requirement is fulfilled. This requirement cannot be solely satisfied by the design.

13. *The user should be able to use the system and understand its interfaces without needing to know how, and with what techniques, the system is implemented.*

None of the user experts noticed any implementation details or had any notion of how the analysis was executed. The only aspect that was noticed were the types of analysis done, i.e., looking at the steps the user takes. This is typical for user workflow analysis. The decision to use user workflow analysis was a separate choice from the techniques, which makes this notion irrelevant for the requirement. In other words, they noticed user workflow analysis was used, but not how this was implemented.

14. *The system should be able to perform the analysis within reasonable time.*

The time the analyses take depends on a lot of things: the PM plugins which have highly differing execution times, the size of the data set, the amount of processing needed, etc. In this research it only took about 5 seconds on a local computer. However, we only developed a small prototype with a small data set, so no real conclusions can be drawn from this. Therefore, we discussed the preferred time range with the user experts.

Two of the user experts immediately mentioned that it was no time-critical application and that it would not be used on a daily basis, so the time it takes does not matter. If it takes too long, they will start it and come back later. The third user expert said that the way of use will be different depending on how long it takes. If it goes very quickly, one will play with it and the different parameters to get to better conclusions. If not, one just wants to specify a lot at the same time, also the same option with different parameters, and come back to it later. However, the third user expert also mentioned that the analysis itself does not need to be real-time, so if the analysis is prepared earlier, for example, a day, that will not invalidate the analysis.

The discussion with the user experts focusses on the production phase, i.e., when the UWAT is used with actual data sets and complete analyses. However, before the UWAT is used, the user workflow analyzers will need to experiment with the analyses to test the analyses and to get a feeling with the options and parameters. In this phase, interaction will probably be needed with the UWAT and the time the UWAT takes should be short. If this is desired, a representative UWAT data set should be prepared which is small enough for the analyses to be performed quickly. In this way, the UWAT and the user workflow analyzers can be prepared for the actual use.

8.2.3 Domain requirements

15. *The input data should have timestamps.*

This statement is about the log data and is out of scope for the UWAT. It is the responsibility of the SUA. The PM analysis can't happen without this timestamp, so this is a hard requirement for the log data. In the logging strategy proposed in this report, this was included. This requirement was fulfilled for the log data used in the prototype.

16. *The output data of the system should be a standardized format that can be used by different visualization tools.*

The output data of the UWAT is a JSON object with the model of the result data set. JSON is a standardized format which could easily be used in different tools. In other words, this requirement is fulfilled.

8.2.4 Usefulness of user workflow analysis

With the user experts, possible alternatives for user workflow analysis were also discussed. An alternative approach for the entire type of analysis that was mentioned often, is standing behind a person and observing what happens or having a conversation or interview with (potential) users. The related risk mentioned for these methods is that the observer can influence the results, especially if they start talking together which is tempting for this complex knowledge. A lack of scalability in these methods is also mentioned. With user workflow analysis, as used in this design, all data can be analyzed instead of just a small subset. It also creates the opportunity to get better and quicker feedback, for example, for new versions of the software.

8.3 Summary

The most important user validation result is that user workflow analysis has added value already for the simple examples implemented in the prototype, and this is expected to become even more when more complex options are implemented. The results were as expected for the specifications by the user experts. The success will mainly depend on the options and parameters available. The user experts also did not feel that the options and parameters restrict them too much. What the user experts can do with the results will also influence the satisfaction. In the validation, the user experts did not notice the implementation details or that the implementation used different tools or techniques.

The system presented was able to use PM and MDE to process a user specification input as a JSON object to an output JSON object which contains the desired results, and which can be used by different visualization tools. The UWAT is capable of processing all the specified patterns to a result that is logical for the user workflow analyzers. The UWAT design fulfills all requirements within its control. However, there are dependencies on external plugins and part of the transformations must be implemented by the SUA owners for each SUA specifically. This can highly influence the fulfillment of requirements in specific implementations.

Due to the generic setup of the metamodels, all kinds of patterns can be extracted and new analyses and functionality of the SUA can easily be added. However, the SUA owner must oversee what these changes mean for the models, transformation and PM execution. In the metamodels, all SUA specifics are limited to the SUA metamodel. In this way, the rest of the metamodels are applicable to all SUAs. Part of the transformations needs to be implemented specifically for each SUA in this design, mainly because the, often SUA-specific, options and parameters play an important role in the transformations. This central role of the user specification makes sure that the specified results, and only these results, are eventually returned to the user workflow analyzer.

9 Discussion

This research presents a design to combine PM and MDE for user workflow analysis of content-intensive applications. A prototype served as a proof of concept to study the added value of user workflow analysis and the combination of PM and MDE. This section discusses the design and validation results. The discussion points are combined into different relevant topics. At the end, the validity is discussed.

9.1 Added value user workflow analysis

In the prototype, some simple examples of user workflow analysis were implemented. These were already experienced as added value, since seeing the results delivered the user experts new insights. An example is the large amount of time spent on the document overview in the Mobina software, while none of the documents of this overview are in the top 10 'Most used processes and documents'. Possible explanations for this are that the document overview is not designed properly or that the content is not clear. Another interesting insight was that only a small part of the subprocesses were used for most processes. This could be caused by some bias due to the small data set, but according to the user experts this could also mean the subprocesses already have too much detail or parts of the reference model are not relevant.

These insights were new for the user experts and therefore clearly show the potential. However, we recommend implementing more examples and discussing these with the user experts to see the added value more complex examples can deliver, to discover what type of analyses offer the most added value, and to give them the ability to discover the real underlying cause for these results.

Based on the initial results, adding user workflow analysis to the toolset of content and software developers seems very useful to get additional insights and eventually improve user experience. However, the added value of the UWAT is strongly correlated with the options that are implemented. Therefore, we recommend that the SUA owner, who is responsible for defining and implementing the options, carefully determines the options that lead to the most added value.

The visualization of the results is not part of the design but is relevant for the perceived usefulness of the analyses for the user workflow analyzers. Possibilities like creating cross-links or interaction with the results are expected to improve the user experience significantly. More about the visualization is discussed in Section 9.2.3.

9.2 Setup of the UWAT

The discussion about the UWAT and its implementation is divided into different subtopics: the combination of PM and MDE, reusability, the separation of visualization and UWAT, user friendliness, and scalability.

9.2.1 Combination of PM and MDE

Our design allows for flexibility in the balance between PM and MDE, supporting a large range of implementations. It allows the implementation to effectively use the strengths of PM, MDE and the implementer to achieve the needs for each option. On the one hand, PM tools can be used only as summarizing tools and the analysis is done through MDE. In this case, the implementer of the UWAT has a lot of control and less dependency on external tools, but the advantages of using PM are not completely utilized and the results depend on the implementation quality. The other extreme is to perform a lot of complex analyses with PM and only use MDE for the visualization. In this case, one can get advanced insights by using the strength of PM and there is less effort for the implementer. However, the implementer also has little control over the outcomes and puts the responsibility in the hands

of external tools. The flexibility to use the entire spectrum between both extremes is provided by the flexible plugin specification and by letting the SUA owner implement the process-to-result transformation.

PM is implemented in this design by external PM plugins. This is a strength of the design since now external knowledge and experience can be used in the tool, something which could never be expected from someone implementing the UWAT. Information from an area that is rapidly evolving now becomes available to a whole (new) range of people. There are also some attention points when using an external tool and, in this case PM. Firstly, the call to the external tool can lead to limitations like the plugin execution taking too long, results that can't be exported, and algorithms that are not implemented yet. The calls to PM can also not (always) be automated yet due to missing APIs. Besides that, one should be aware that one has limited control over the behavior of the plugins. In this case, this may have consequences for data integrity and deterministic behavior, but it can also, for example, have undesired behavior related to privacy and security. Finally, the design expects the external call to PM to deliver a model compliant to the PM metamodel. This may not be possible yet.

However, these remarks do not mean that PM should not be used. PM has a lot of added value and contains advanced techniques, which are now accessible to every SUA owner. PM is an area that is rapidly evolving, so many of the limitations are expected to become less in the future. Additionally, the SUA owner also has a lot of flexibility because you can easily swap one plugin with another one (e.g., because another, newer, plugin has more benefits). We strongly recommend using PM, and only present these attention points so one can create the best solution and has no surprises when implementing PM.

Due to the standardization using MDE, it becomes easier to benefit from PM. The PM plugins only need to be researched and implemented once, and then all potential user workflow analyzers can use it.

9.2.2 Reusability

One of the goals in the design was to create a reusable application, i.e., to create an UWAT which can be used by other SUAs as much as possible. This was achieved to a high extent. All (meta)models are setup generically, which means that their design is independent of which SUA is implemented, except for the SUA metamodel. The SUA metamodel is developed specifically for each SUA and gives SUA owners the possibility to add SUA specifics to the analyses. Due to this setup, the SUA owner only has to develop this metamodel, and is not burdened with the other metamodels. After the initial set-up, it only needs to be adapted if extra SUA functionality needs to be analyzed. The effort is therefore very low and is not expected to create a problem. An interesting future research direction would be to develop a metamodel for the SUA metamodel. Certain concepts may be applicable to all SUAs, which can be represented in the metamodel. This could be a nice guideline for SUA owners on what to put in their SUA metamodel, but it is also a first step towards a standardized UWAT leading to even less development effort for the SUA.

In this design, we give the SUA owners a lot of flexibility in which analyses to implement instead of limiting it to a predefined set. The core of the implementation of the UWAT are the options which the user workflow analyzers can select. Because these options are often SUA-specific and will be defined by the SUA owner, the preparation-to-specification transformation and the process-to-result transformation need to be implemented by the SUA owner to a large extent. The latter transformation also provides the opportunity to include the SUA specifics in the result. The result-to-dashboard transformation is generic and has no SUA specifics, thanks to the usage of generic result types and the standardized format of the output JSON object.

Even though the flexibility in option and parameter definition makes the preparation-to-specification and process-to-result transformation not completely standardized, this research shows that some parts can be standardized in the transformation definition and implementation. For example, all relations to the SUA model in the process-to-result transformation of the prototype are implemented in helper methods. Additionally, the plugin definition is separated from the user specification which improves reusability, since the same plugin definition can be shared amongst a lot of specifications. Also, the preparation-to-specification transformation contains some reusable parts, because the input JSON object has a standardized format. These are already first steps towards standardization. In future research, it will be very interesting to explore this with more complex options. An interesting topic would also be to standardize frequently asked analyses, e.g., frequency, since it can be applied to many different objects.

To offer the desired flexibility, the SUA owner has a lot of responsibility in this design. The SUA owner should make a final selection of options and parameters, based on the business objectives. Next to that, he/she is also responsible for implementing these options and determining what they mean for PM. This demands a high conceptual level and knowledge of PM, something that may not be available at each SUA owner. However, since the SUA is also developed in a software company a certain level of expertise is probably available. Additionally, only the UWAT implementer needs to have this expertise, while a whole range of other people can benefit from it. In the future, more advanced options can be researched like using a library of PM plugins and implementation examples, or standardizing options which can be shared amongst SUAs. A SUA metamamodel can also help in this. In this way, less responsibility is put on the SUA owner, making the UWAT more accessible to all SUAs. For this research, it was out of scope.

9.2.3 Separation visualization and UWAT

A clear decision was made to separate visualization from the implementation in the architecture to allow for flexibility in visualization. The UWAT only takes a JSON object as input and also provides a JSON object as output. The way these data is created or visualized respectively, is left up to the SUA owner. The greatest advantage of this decision is that it gives great flexibility to the SUA owner. He/she can now visualize the result in the preferred way for their SUA. If the SUA owner would like to do post-processing or use the data in their own dashboard, this is possible. The result model is set up in a way that it can directly be used for visualization, hence the structured formats and the naming possibilities. However, this makes it less suitable for post-processing at some points. For example, all attributes of the content names are already transformed into a combined string for visualization. To extract the attributes, you need to parse this string. This was a design choice, to put as little as possible burden on the SUA owner. However, if post-processing by other tools, e.g., BI tools or other visualization tools, is preferred, this should be reconsidered.

The user expert feedback also brought up that users would like to interact with the results. This gives the user workflow analyzers the flexibility to test hypotheses and get additional insights. To implement interactive behavior, this has impact on either the UWAT or the visualization. If this is added to the visualization, this means that the SUA owner has all the flexibility to do it his way. However, this contradicts the philosophy behind the design to put all the intelligence in the process-to-result transformation. This might also need adjustments to the result metamodel, since additional semantic information is possibly needed to offer this interactivity. On the other hand, the result model can include results for multiple options at once allowing the user to switch between these results in an interactive way. This enables interactivity, but also means that the interactive options have to be defined and implemented upfront. This kind of behavior may also put additional requirements on the responsiveness of the UWAT. We recommend experimenting with the need for this with some (complex) use

cases in the next design cycle, to see how the user workflow analyzers and the SUA owner can best be served.

9.2.4 User friendliness

An important objective of the design was the user friendliness, i.e., that the user workflow analyzers should not be bothered with the techniques and implementation of the UWAT. The design presented here successfully implements this objective. In the validation, the user experts did not notice anything of the implementation. The system is presented, and experienced by the user experts, as one system, even though multiple external calls are necessary. The design is built in a way that the UWAT handles all external calls, and the user workflow analyzers are not bothered with this, i.e., whether this is implemented using PM, MDE, or other technologies makes no difference. Therefore, also no user involvement is necessary after the user specification.

This design has as a foundation that options and its parameters are predefined and that the user workflow analyzer can only select these options. This is because every option has deep impact on what to do using PM and how to process this using MDE; the user cannot freely define options. The SUA owner predefines the options based on business objectives, to ensure that the options deliver added value to the user workflow analyzers. However, the options could potentially be experienced as limiting by the user workflow analyzer. This was discussed with the user experts, but this was not seen as a limitation, as the options and parameters are experienced as useful. All three believe that these options are necessary due to the complex analysis happening within the UWAT. In conclusion, as long as the SUA owner defines interesting and useful options, the options and parameter definitions are a solid way of implementing potential analyses.

We assumed that the role of the SUA is important to create a user-friendly solution. In other words, they should feel like the analysis is about the SUA. In our design, the SUA metamodel brings the opportunity to add SUA specifics, which are embedded in the result in the process-to-result transformation. In the implementation of the prototype we used the names of the content elements in the process-to-result transformations. This gave the user experts the feeling that the analyses were about Mobina. The options and parameters also gave this feeling, especially since one of the parameters was about a core data structure of the Mobina content. Thus, this case study showed that involving the SUA does indeed improve the experience of the user workflow analyzers and should be included by the SUA owners when implementing the UWAT.

9.2.5 Scalability

An important reason to use user workflow analysis is its potential to become scalable. Compared to other methods, this method stays useful when new data or functionality arrives, and little extra costs must be made. The design is set up in such a way that it is flexible. New functionality of the SUA can easily be added to the SUA (meta)model. To use this functionality, new options must be introduced, or older options must be updated. These changes must be implemented but can easily be built on top of the other analyses without influencing them. This also brings the flexibility to adapt to newly requested analyses or changes from the user workflow analyzer. New data, keeping versioning in mind, can also easily be added and just as well be processed by PM. In this way, the solution is scalable, which was also recognized by the user experts. They were asked about other methods to achieve similar results, but only different forms of user testing were mentioned. They mentioned these scaled badly. Besides that, another disadvantage was mentioned for other methods: the risk of influencing these results. This solution also diminishes that risk.

In this research, we looked at scalability from the perspective of being flexible when new data, functionality or analyses arrive. However, scalability is also about handling large amounts of data and complex implementations. This design was only implemented for very simple analyses and on a small data set. For this proof of concept, testing additional scalability aspects was out of scope. We however recommend larger scale tests in the next stage of the design to test this aspect of scalability.

9.3 Implications of proof of concept

We presented a basic design since it is still in the first stage of the design cycle. It contains all the important core concepts, but it does not take into account all details and practical implications. Versioning, and SUA specifics that could influence the log data and the results, like permissions or configuration of the content, are for example not yet accounted for. However, thanks to the setup of our metamodels and transformation, all these characteristics can be built on top of the current design; it does not influence the core of the design. Therefore, this was out of scope for this first stage of the design cycle. It is very interesting to explore these aspects in the next stage.

In this research, a prototype of the design is implemented. This already showed some nice possibilities for the actual implementation, as well as research possibilities. Firstly, the implementation was made efficiently because only the selected options were analyzed. No unnecessary processing was done. This could have positive consequences for the scalability in the actual implementation. Secondly, the prototype was not yet fully automated due to the manual intervention. This is undesired behavior for the real application. However, preparing the PM also led to a nice opportunity. If the most recent data is not crucial, part of the results can be cached to make the total implementation faster. Most time reduction can be gained at PM since some of the plugins take very long, but caching can also prove useful for the UWAT itself when using large data sets.

One should be aware of the data integrity risks in this design due to the many data conversions that take place. For the small examples implemented in the prototype, we could ensure data integrity by carefully looking at the implementation and validating the result of the test data set. However, this can be compromised in larger and more complex transformations, as well as by bad code quality or external implementation flaws. We therefore recommend using additional checks and tests on this aspect when implementing the UWAT to ensure no undesired behavior happens.

Since this was only a prototype implementation, some aspects were out of scope. However, these might still be important when implementing the SUA commercially. Firstly, this was only tested for a very small data set (25 sessions of in total app. 7,5 hours). Performance of both the UWAT and PM plugins could reduce with larger data sets. Secondly, there are several organizational aspects like how to handle the privacy of people in the data. It should be communicated upfront that these data are collected and how this is going to be used in analyses. Finally, the PM implementation limits full automation now. If PM can still not be automated in the nearby future, a real solution must be found for this. Implementing some sort of middleware to handle the PM execution and to process the results is an interesting topic when looking into possibilities to fill this gap.

9.4 Validity

This research was set up as a proof of concept. Since it was the first stage of the design cycle, the validation was setup in a way that it gave a good first indication of the possibilities of the UWAT and its design and that it explores the strengths and weaknesses. However, this also causes some threats to the validity of the conclusions. This subsection discusses these aspects that need to be validated more thoroughly.

Firstly, the UWAT is now only implemented for very simple examples. A few examples were selected of which was known that PM could support them well and that could be analyzed with the current data set. This gave a good first indication about the added value of user workflow analysis implemented using PM and MDE. However, for other type of analyses the approach may not be as useful. Additionally, the architecture may not handle more complex examples properly. For now, the added value of the UWAT in complex analyses is based on a thought experiment with the user experts. However, the actual results can maybe not be overseen. We recommend implementing and validating the UWAT with more complex examples and a complete question set for a potential SUA to validate which types of analyses it supports.

The UWAT is now only implemented and validated for one case study. Different case studies may have different questions and different requirements. Therefore, standardization is now only discussed from a theoretical perspective. To have a quantitative analysis on how reusable the architecture is, the UWAT should be tested for multiple applications.

Additionally, only a small set of potential user workflow analyzers were used in this validation: the content developers. These user experts were the only ones suitable to draw conclusions about the results in this case study. Software developers or potential other types of users may look different at what they want to achieve and what the desired outcomes are. All content developers also had an academic background, which in general leads to a more conceptual level of thinking, and they are used to working on the edge of business and IT, which means that they are familiar with systems like the UWAT. Furthermore, all user experts were from the same organization. All these characteristics could however lead to a bias in the validation results.

Fourthly, only a prototype was implemented. Manual intervention for PM was allowed and simplified views were used like only looking at the entire data set. However, when the SUA evolves, different requirements can arise, and unexpected problems can come up. For example, how to handle functionality that is removed. This was out of scope for this research but should be validated in next stages of the design cycle.

The technical review can also be a potential threat to validity. Several aspects were tested by reasoning about the design and the implementation. However, no experiments or tests were executed yet, because only very simple examples and a prototype implementation were available. Certain aspects like scalability and performance can only be tested properly by running experiments. These are important additions to properly validate the design.

The technical review was performed by the researcher. The researcher can be biased because of her enthusiasm or pessimism about the design. The researcher is also completely familiar with the design and the design choices. In this stage, these insights were necessary for the validation, because many concepts were still abstract. However, the researcher can have blind spots. To exclude this potential bias or blind spots, technical experts which were not involved in the design phase should be asked to validate the design.

Finally, the researcher was in this case very familiar with the SUA and therefore also had the role of SUA owner. This was convenient to make quick progress. However, this also means the researcher may not always have been completely objective about the outcomes and/or the validation. Additionally, the researcher also knew the user experts. Therefore, the user experts might have given desired answers rather than critical answers, influencing the interview results.

10 Conclusion

This study shows that user workflow analysis can successfully be implemented by combining the strength of Process Mining (PM) in extracting patterns from event log data and the strength of Model-Driven Engineering (MDE) in standardized and automatic transformations. We presented a User Workflow Analysis Tool (UWAT) which user workflow analyzers (i.e., content and software developers) can use to get additional insights in their content-intensive application, referred to as System Under Analysis (SUA), in a standardized, flexible, and user-friendly way.

This chapter first presents a summary of the report and the answers on the research questions. Then, we present the implications and recommendations for content-intensive applications. Finally, we present our contribution to science. In this chapter, also interesting topics to explore in future work are mentioned.

10.1 Summary

Traditional methods to evaluate software are often not suitable for content-intensive applications due to their inability to evaluate the behavior of users across the entire application, because they are not scalable and because the people used in the test often do not have the time to grasp the often complex content. In this research, we presented user workflow analysis as a solution for these shortcomings. User workflow analysis helps content and software developers of content-intensive applications to gain (additional) insights in the behavior of the users and the effects of their work because of its ability to track the actual steps of the user. Because it looks at the objective log data that includes all the different steps the user takes, user workflow analysis has a broader scope than the details of an interface, tracks behavior across the entire application and enables to draw conclusions without interference of the researcher or the artificial setting. Due to these advantages new insights can be gathered which is not possible in traditional methods like user testing. This research shows that user workflow analysis has a lot of potential added value for improving content-intensive applications.

We presented a design which uses PM and MDE to implement user workflow analysis. This design presents a solution which can be fully automated, as soon as the necessary PM support is there. A flexible solution is created where new functionality and analyses can easily be added. Therefore, this design mitigates the shortcoming of traditional methods by being more scalable and thus suitable for continuously improving the software, for example easily observing how new versions of the content and/or software changes the experience of the user.

PM is used to extract patterns from the event log data that are useful for the user workflow analysis in this design. External tools can be used so the organization can benefit from the experience of others. Thanks to the separation of the plugin specification from the analyses defined for each SUA, other PM plugins can be easily selected when a better plugin becomes available. PM tools however often have the disadvantage that it is left up to the users to interpret the results and do potential post-processing on the results. This expects a lot of knowledge about PM from the user of the tool. Every time manual intervention is needed to perform the necessary actions. These shortcomings of many PM implementations make PM less usable in many cases.

In this research, MDE is introduced to mitigate these shortcomings. MDE is a powerful technique to standardize information and translate and model the information on an abstract and more generic level. Before this research, this combination had not been extensively researched and no references were found where it was used for configuring PM and analyzing the results. In this research, we presented a design which can not only automatically include PM in the analysis, it also offers the opportunity to standardize

analyses. In this way, only the implementer needs to specify the PM plugins needed for the different analyses and implement the processing of their outcomes. All other potential user workflow analyzers can benefit from these analyses in a user-friendly way without being bothered with the technical implementation. Next to this, the MDE adaptation makes, as soon as the PM execution can be called automatically, the solution fully automatable so no human intervention is needed anymore after the initial implementation.

The presented design is to a high extent reusable for different content-intensive applications. All metamodels are generic for each SUA, except for the SUA metamodel. Each SUA must implement a part of the transformations, giving the SUA owner flexibility and the ability to define their own options. The other parts of the transformations are generic and can easily be reused. Besides that, the PM plugin specification is separated from the specification of the user workflow analyzer, so not every SUA owner needs to reinvent the wheel when it comes to PM. This generalization creates a balance between giving the SUA enough flexibility to implement their own analyses on the one hand, and on the other hand giving them the benefits of reusing proven concepts.

10.2 Implications and recommendations for content-intensive applications

User workflow analysis has a lot of added value for content and software developers of content-intensive applications. Content and software developers gain additional insights in how the content and software is used. They can use these insights to improve the software and its content which can lead to a better user experience. We would recommend to content-intensive applications like Mobina to extend their toolset with user workflow analysis to improve the experience of their users and to let content and software developers get a grip on their work and the effects on the users.

During the validation, the most positive aspects mentioned by potential user workflow analyzers are the scalability and the objectivity of the approach. The analyses and settings available in the UWAT are important for the satisfaction and the experienced added value. Simple examples, as implemented in the prototype, already have added value. Options that are more related to the (structure of) the software and content provide even more added value. By selecting the correct options, content and software developers are expected to greatly benefit from the UWAT.

The set-up of the design has some great advantages for the implementer of the UWAT in the organization. Due to the generic set-up, many parts of the UWAT can easily be reused for each SUA. The approach also provides a lot of flexibility. SUA owners have to implement part of the transformations themselves, allowing them to control which analyses can be executed and giving the flexibility to define their own options instead of being limited to a predefined set. Additionally, new functionality and analyses can easily be added. One is also flexible in the amount of PM and MDE to use in the implementation, i.e., for what to use external expertise with PM and what to implement themselves with MDE. The preferences and capacities of the implementer can be taken into account for this.

The solution presented gives organizations the opportunity to use external knowledge and expertise thanks to the abstraction through MDE. There is a lot of research happening in the area of PM and techniques and implementation are rapidly evolving. Using this design, this knowledge and development can easily be used. When better algorithms and techniques become available, these can also be easily integrated in the analysis, making the solution future-proof.

The visualization is separated from the implementation in the design, so one can use a tool that is familiar to the user workflow analyzers or that is integrated in the development environment of the SUA to visualize the result and the set-up of the analyses. A request that came up during the validation with potential user workflow analyzers was that interaction with the results would be very interesting. This is a nice opportunity that should be explored in future work.

Finally, most people that benefit from the UWAT are not involved with the technical implementation or the characteristics of the design and its implementation. Only the implementer needs to put the effort in implementing the analyses and selecting the correct PM plugins, and afterwards all user workflow analyzers can benefit from user workflow analysis in an approachable way. It makes this way of analyzing the software use available to a whole new range of people including the people who have no experience in developing and evaluating software.

10.3 Scientific contributions and future research

This research successfully shows how two techniques can be combined and how the strength of one technique can mitigate the shortcomings of another technique. The strength of standardization and generalization of MDE could perfectly fill the gap left by PM, namely the configuration of the plugins and the interpretation and post-processing of the outcomes. This shows that two powerful techniques can lead to an even more powerful combination.

For every combination of techniques, you need to consider to what extent to use each technique. These techniques have an overlap in analyses they can both perform. It leads to a discussion about what to build yourself, i.e., in the UWAT using MDE, and for what to rely on external knowledge, i.e., using PM plugins. The design of this research has its strength in enabling both possibilities and giving all the flexibility to the implementer. When selecting techniques to combine, one should be aware that there could be a tradeoff between both techniques.

We learnt several things about the techniques selected in this research. First, MDE is known for its standardization and abstraction. This strength clearly came forward in this research, due to our setup of the metamodels and transformations. MDE greatly empowered us to create a flexible and reusable solution. Especially in use cases where this is desired, we recommend using MDE. As an addition to this, we noticed that the selection of techniques affects your solution. This is often in a positive way as with MDE, but it can also introduce a bias in the selection of use cases. For example, because we used PM, analyses that benefit from these techniques were selected. These are clearly experienced as added value, so this is a good starting point for user workflow analysis. In future work, it might be very interesting to also explore other techniques, e.g., machine learning, to enrich the analyses and compare the strengths of even more techniques.

Independently of the design and the implementation, this study shows that user workflow analysis is a promising area. Only the simple examples implemented already provided added value. We therefore recommend to keep exploring this topic, since it can help a lot of people to gather insights in the application. In this research we only considered content-intensive applications. It would be very interesting to explore other areas of applications in future work to see whether and how they can benefit from user workflow analysis and the design presented in this thesis.

The design presented in this thesis is a basic design. The strength of the design is that it embodies every core aspect that is hard to adapt later, like reusability and flexibility. On top of this stable design, people can build extra aspects like versioning. We successfully distinguished in this study core aspects, which were crucial in the design, from details which can be added later when the first design is validated positively and is ready for the next design cycle. In future work it is interesting to see how this basic design can be enriched with other aspects that are more detailed or application-specific like versioning or content configuration.

Since this was only an early design phase, the design and implementation were only validated with a small data set and for one SUA. It will be very interesting to explore how the design handles actual data sets which are evolving, and which are of a larger size, to properly test the scalability and to see how the UWAT will work when completely implemented. Additionally, it would also be interesting to implement it for multiple SUAs to get a feeling to what extent the design is reusable, but also to get an initial idea on how many similarities there are in what to analyze and which business objectives to answer. This could shine a light on a lot of interesting analyses for the future.

Unfortunately, there is still a gap between design and implementation. For example, our design proposes a fully automated solution where the UWAT can call the PM execution without being bothered with the specifics of the PM implementation and outcomes. In practice, not all PM executions can be called automatically. It is important to be aware of this gap and when moving to the next step in the design cycle, to consider these aspects and propose a way to fill this gap. We strongly recommend keeping the design as is and not removing the external call to PM, since then you have the best solution when all implementation parts are available. Therefore, in this case we would recommend in future work to explore some sort of middleware which can be called by the UWAT and which can execute the PM plugins and process the result in the correct way. This could also bring great benefits to the PM research area, independently of their added value to the UWAT, since it will allow for combinations of techniques, integration of tools and automatic and standardized processing of the results.

We learned that it is not possible to validate all requirements based on design. Several of the requirements depend on the implementation, i.e., the implementation determined whether the requirement was fulfilled. Since the implementation can be different for each SUA, we cannot establish that an UWAT based on this design will automatically fulfill all requirements. This is not necessarily a problem, but one should be aware of this.

Finally, it would be very interesting to explore ways to let different SUAs reuse parts. This can be on many different levels. A first step is to share parts of the implementation. Good starting points for this are to share the PM plugin specifications with all the SUAs, or to create a metametamodel for the SUA metamodel as a starting point for the SUA owner and to standardize more of the transformations. A next step could be that entire analyses are shared. Then, only the custom data set and SUA specifics must be added and for the rest a standard analysis can be used. Eventually this could even lead to a party having an UWAT-as-a-Service, where a third party implements the analyses and all SUAs can use these analyses. In this way, the effort and complexity are limited to one party, and all different SUAs can benefit from this.

References

- [1] International Standards Organization, "ISO 9241-210:2010 Ergonomics of human-system interaction -- Part 210: Human-centred design for interactive systems." International Standards Organization, 2010.
- [2] K. Hornbæk, "Current practice in measuring usability: Challenges to usability studies and research," *Int. J. Hum. Comput. Stud.*, vol. 64, no. 2, 2006.
- [3] M. Maguire, "Methods to support human-centred design," *Int. J. Hum. Comput. Stud.*, vol. 55, no. 4, 2001.
- [4] A. Abran, A. Khelifi, W. Suryn, and A. Seffah, "Usability meanings and interpretations in ISO standards," *Softw. Qual. J.*, vol. 11, no. 4, pp. 325–338, 2003.
- [5] H. X. Lin, Y.-Y. Choong, and G. Salvendy, "A proposed index of usability: A method for comparing the relative usability of different software systems," *Behav. Inf. Technol.*, vol. 16, no. 4–5, 1997.
- [6] S. Kujala, "Effective user involvement in product development by improving the analysis of user needs," *Behav. Inf. Technol.*, 2008.
- [7] J. J. Baroudi, M. H. Olson, and B. Ives, "An Empirical Study of the Impact of User Involvement on System Usage and Information Satisfaction," *Commun. ACM - MIT Press Sci. Comput. Ser.*, 1986.
- [8] F. D. Davis and V. Venkatesh, "Toward preprototype user acceptance testing of new information systems: implications for software project management," *Eng. Manag. IEEE Trans.*, 2004.
- [9] W. Hwang and G. Salvendy, "Number of people required for usability evaluation," *Commun. ACM*, 2010.
- [10] S. Kent, *Model driven engineering*, vol. 2335. 2002.
- [11] Object Management Group, "MDA Guide rev. 2.0," 2014.
- [12] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, 2nd ed. Morgan & Claypool, 2012.
- [13] A. M. Elswawi, S. Sahibuddin, and R. Ibrahim, "Model driven architecture a review of current literature," *J. Theor. Appl. Inf. Technol.*, vol. 79, no. 1, pp. 122–127, 2015.
- [14] J. Whittle, J. Hutchinson, and M. Rouncefield, "The state of practice in model-driven engineering," *IEEE Softw.*, vol. 31, no. 3, pp. 79–85, 2014.
- [15] J. Bézivin, "On the unification power of models," *Softw. Syst. Model.*, vol. 4, no. 2, pp. 171–188, 2005.
- [16] P. Mohagheghi and V. Dehlen, *Where ss the proof? - A review of experiences from applying MDE in industry*, vol. 5095 LNCS. 2008.
- [17] R. F. Paige, N. Matragkas, and L. M. Rose, "Evolving models in Model-Driven Engineering: State-of-the-art and future challenges," *J. Syst. Softw.*, vol. 111, pp. 272–280, 2016.
- [18] D. S. Kolovos *et al.*, "A research roadmap towards achieving scalability in model driven engineering," in *ACM International Conference Proceeding Series*, 2013.
- [19] W. Van Der Aalst *et al.*, *Process mining manifesto*, vol. 99 LNBIP, no. PART 1. 2012.
- [20] W. M. P. Van Der Aalst and S. Dustdar, "Process mining put into context," *IEEE Internet Comput.*, vol. 16, no. 1, 2012.
- [21] W. van der Aalst, *Process Mining, Data Science in Action*, Second edi. Berlin: Springer, 2016.
- [22] W. Van Der Aalst, "Process mining: Overview and opportunities," *ACM Trans. Manag. Inf. Syst.*, vol. 3, no. 2, 2012.
- [23] B. Keith and V. Vega, *Process mining applications in software engineering*, vol. 537. 2017.
- [24] W. M. P. van der Aalst, "Business alignment: Using process mining as a tool for Delta analysis and conformance testing," *Requir. Eng.*, vol. 10, no. 3, 2005.
- [25] W. Poncin, A. Serebrenik, and M. Van Den Brand, "Process mining software repositories," in *Proceedings of the European Conference on Software Maintenance*

- and *Reengineering*, CSMR, 2011.
- [26] W. Van Der Aalst, “Big software on the run: In vivo software analytics based on process mining (Keynote),” in *ACM International Conference Proceeding Series*, 2015, vol. 24–26–Augu.
- [27] V. A. Rubin, A. A. Mitsyuk, I. A. Lomazova, and W. M. P. Van Der Aalst, “Process mining can be applied to software too!,” in *International Symposium on Empirical Software Engineering and Measurement*, 2014.
- [28] I. Ailenei, A. Rozinat, A. Eckert, and W. M. P. Van Der Aalst, *Definition and validation of process mining use cases*, vol. 99 LNBIP, no. PART 1. 2012.
- [29] W. M. P. Van Der Aalst, *Business process simulation revisited*, vol. 63 LNBIP. 2010.
- [30] M. Song and W. M. P. Van Der Aalst, “Supporting process mining by showing events at a glance,” in *WITS 2007 - Proceedings, 17th Annual Workshop on Information Technologies and Systems*, 2007, pp. 140–145.
- [31] J. Simonin, J. Soulas, and P. Lenca, “Activity Monitoring Process based on Model-Driven Engineering-Application to Ambient Assisted Living,” *J. Intell. Syst.*, vol. 24, no. 3, pp. 371–382, 2015.
- [32] S. Bernardi, R. P. Alastu y, and R. Trillo-Lado, “Using process mining and model-driven engineering to enhance security of web information systems,” in *Proceedings - 2nd IEEE European Symposium on Security and Privacy Workshops, EuroS and PW 2017*, 2017, pp. 160–166.
- [33] A. Mazak and M. Wimmer, “On marrying model-driven engineering and process mining: A case study in execution-based model profiling,” in *CEUR Workshop Proceedings*, 2016, vol. 1757, pp. 78–88.
- [34] A. Mazak, M. Wimmer, and P. Patsuk-B sch, *Execution-based model profiling*, vol. 307. 2018.
- [35] I. Sommerville, *Software engineering*, 7th ed. Addison-Wesley, 2004.
- [36] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*, 1st ed. Wiley Publishing, 1998.
- [37] TUe, “ProM.” [Online]. Available: www.promtools.org. [Accessed: 08-Jan-2018].
- [38] Fingent, “Off-the-shelf vs. Custom Software : Making the Right Choice for Your Business.” [Online]. Available: <https://www.fingent.com/blog/off-shelf-vs-custom-software-making-right-choice-business>. [Accessed: 12-Jul-2018].
- [39] PCD, “The Pros and Cons of Custom Software vs. Off-the-Shelf Solutions.” [Online]. Available: <http://pcdgroup.com/the-pros-and-cons-of-custom-software-vs-off-the-shelf-solutions/>. [Accessed: 12-Jul-2018].
- [40] I. Sommerville, *Software Engineering*. 2010.
- [41] E. S. De Almeida *et al.*, *The domain analysis concept revisited: A practical approach*, vol. 4039 LNCS. 2006.
- [42] J. Li, M. Torchiano, R. Conradi, O. P. N. Slyngstad, and C. Bunse, *A state-of-the-practice survey of off-the-shelf component-based development processes*, vol. 4039 LNCS. 2006.
- [43] X. Zhang and H. Pham, “Software field failure rate prediction before software deployment,” *J. Syst. Softw.*, vol. 79, no. 3, pp. 291–300, 2006.
- [44] C. Wohlin, P. Runeson, M. H st, M. C. Ohlsson, B. Regnell, and A. Wessl n, *Experimentation in software engineering*. 2012.
- [45] R. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. 2014.

Appendix A. Business objectives Mobina

Main objectives for content developers:

1. Is it clear what we mean?
 - a. Do they read the description?
 - b. Do they open the information tab?
 - c. How long do they stay at the same model element?
 - d. Do they come back to the same place?
2. Can the user find what he/she is looking for?
 - a. What is the path the user follows?
 - b. How do they come to a certain place?
3. Does it trigger action?
4. Is it converging?
5. Completeness of the model?
 - a. Is there too much in the model?
 - b. Is there too little in the model?
 - c. What subprocesses or documents are used?
 - d. What process-document relations are used most often?
6. Who uses what?
 - a. What roles go to what processes/documents?

Main objectives for software developers:

7. Is the reference model functionality correctly structured?
 - a. Are the tabs in the correct order?
 - b. Is the description on top useful?
 - c. Is the information correctly structured and separated?
 - d. How often are certain functionalities/buttons used?
 - e. How do they go through a reference model?
 - f. What is the entry point of the model element?
 - g. Do they go directly to sub processes or first click the process itself?
 - h. What are the number of steps/clicks necessary to reach the destination?
8. What leads to action with the user?
 - a. What do people do before they place a comment?
 - b. What do people do before they give a rating?
 - c. What do people do before they enter a keep or improve?
 - d. What do people do when they get tagged?
 - e. What kind of users react to each other?
9. How performs the help functionality within the software?
 - a. What is the usage of help buttons and hovers?
 - b. How much help is needed for functionalities/per tab?
 - c. Where is the most help needed?
 - d. What are the actions after the use of the help button?
10. What leads to a decision for the critical aspects?
 - a. How long does the decision take?
 - b. What actions does the user do before making a decision?
11. What is the added value of the supporting functionality in the collaboration environment?
 - a. What tags are used?
 - b. What types of tags are used?
 - c. Are more 'I agree' or 'I disagrees' given?
12. Is the separation between Keep-Improve and Ratings clear?
 - a. Do people use both?
 - b. Do people collaborate within the keep and improves?
 - c. Do people often switch between those?

13. How we can support the user after login?
 - a. What are the first actions after login?
 - b. Do people often go the same model elements in different sessions?
 - c. What menu items are used?

Main objectives for both content and software developers:

14. Is the reference model correctly structured (in the software)?
 - a. Do people use the directory often?
 - b. Do people use both processes and documents?
 - c. Do people go the processes in earlier sessions and then to documents in later sessions?
 - d. Do people come together in the documents?
 - e. Do people jump more over levels or more over documents?
 - f. Do people go back and forth between model elements/in the directory?
 - g. Do they look at the related processes of the documents?
 - h. Do people go directly to the document overview from the menu?

Appendix B. Data collection document for the participants

Data collection Master Thesis Marlène Hol

This document is part of the data collection phase of the Master Thesis of me, Marlène Hol. I am a master student Computer Science at the University of Twente. This research is conducted for Mobina, where I am also Co-Founder and Vice President of. The research will be used by Mobina to analyze the user workflow and to learn lessons about the content and software structure.

In this document I will shortly explain the research and how this step is part of this research. Next, I will give a small introduction into the most important parts of the software. Finally, I will introduce what is expected of you today. This also includes the assignments I will ask you to do. The log data from these assignments will be used in the research.

The project

Over the past years, the role of the user in software development got more attention. Human-centered design and involving the user of the development became a standard. However, there are two large disadvantages about many of these (traditional) methods. First, these methods often focus on the user interface and details of the design. Of course, this is important and useful, but it makes them less usable for determining whether the software and its content are structured correctly and what the actions are across screens. This is especially important in knowledge-intensive applications. Secondly, these methods are often used for the initial design and not for continuously improving the software, because of their lack of scalability. However, learning from the actual behavior is of course from high value for the quality of the software and its content. For these two reasons I will research an upcoming way of evaluating software: user workflow analysis. The user workflow represents the different steps or tasks the user is taking across screens in the artifact, in this case the software application. This can be for a specific goal, e.g. to reach a certain state or perform a specific action, or a sequence of steps with no particular purpose in mind. For this research, I will use two specific techniques for user workflow analysis: process mining and model-driven engineering.

The research questions of this research are:

1. Can user workflow analysis help content and software developers of knowledge-intensive applications to gain insights and information about the content and the software is a manageable, standardized, and automated way?
2. Can user workflow analysis implemented by the techniques process mining and model-driven engineering be used to reach this goal?

To answer these research questions, a case study will be used, in this case the knowledge-intensive application Mobina¹. Input of this research will be the log data. All steps the user takes in Mobina will be logged, which will be input for the process mining. With process mining the process of the user can be extracted from the log data. The generated process models can then be used as input for model-driven engineering. Model-driven engineering makes it possible to translate the process models to interesting information and relations for the content and software developers, using standardized models. The final step will be that from these models a dashboard can be generated for the content and software developers.

¹ www.mobina.it.com

The product

In this section I will introduce the most important concepts and functionality of Mobina. Note that this is only a description of the functionality and not of the content. Information about the necessary content can be found in the application itself. Also note that this does not include all the functionality of Mobina. This research is scoped to the reference model, so other functionality is out of scope.

When you login, you will directly go to the reference model. If you select a process or document, you will see different tabs. The processes and documents have the exact same functionality and therefore the exact same tabs. I will explain each of the tabs shortly.

Reference model

In this tab you see the different processes and document. In Mobina, processes are actions that are executed within a manufacturing company. Documents are the information exchanges between the processes. Both processes and document know a break-down structure. The documents can be linked to processes on all different layers.

When you are at a process, you see the processes on the left. This can be the sub processes or the same layer processes. You can see this by the directory (whether it includes sub processes) or by the face that on the same layer the selected process is dark blue. On the right you see the related documents. If you click on a document, you can also see the processes the document is linked to. If you are at a document, you see a similar view. You then see the processes where the document is linked to on the left, and the subdocuments on the right.

Information

In this tab, you can download attachments related to that process or document, see the critical aspects that are linked to the process or document, and see the innovations that are linked to the process or document. At the moment of data collection, there are no critical aspects or innovations linked to the processes and documents yet. On some places you can find attachments.

Discussion

Here you can find the discussion possibilities. You can have online conversations here with other employees. You can start your own conversations or reply to existing conversations. You can also tag other employees if you want their attention or tag an IT system of your company. Finally, you can use the arrows to agree or disagree with a comment.

Keep-Improve

This functionality can be used to add things you want to keep and things you want to improve for the IT landscape. You can do this for the entire IT landscape or for a specific IT system. You can also use the arrows here to agree or disagree with someone else's keep or improve and give a short explanation why you agree or disagree. Note that you are working together on this list of keeps and improves, so other people can see what you add.

Ratings

With the ratings you can give your own rating for the IT landscape. Again, you can do this per IT system or for the entire application landscape. You can give a rating between 1 and 7 and give an explanation for your rating. Note that you do this individually, so other people won't see your rating, only managers.

Analysis

In this tab, you can see some analysis of the process or document. You can find information here about the progress, but also about what is popular and not popular for that process or document.

If you have any questions about the software, feel free to ask me.

Assignments

In this section I will introduce what is expected of you today. I will also introduce the five assignments here. Important to mention is that I am present during the execution of the assignments to check the technology, solve potential problems, and to answer questions about the software. However, I will not be there to answer questions about the assignments. This is because the actual users of Mobina will also not have this option either and if I would interact with the assignments, that data will not reflect real use of the software anymore.

Goal

You will be one of the five data collectors of this research. The goal of the data collection is to create log data which is representative for the real use of Mobina. Because we want to trigger different behaviors and to ensure that different kind of questions can be answered, a set of five assignments is created. You are asked to do all five assignments, which can be found in the next section. Each of these assignments is a small scenario where you are a different person in a manufacturing company, Smart Crane. For each of the scenarios, you will use the Mobina software to improve your process, for example by sharing your issues or to motivate other employees. Each of the scenarios explain the goal and necessary information, however there is no method description. This means you have a lot of freedom on how to fulfill the assignment. This is done on purpose to reflect real use and to learn from the experiences. Important to mention here is that this also means that there are no wrong methods or ways of fulfilling the assignments. Just try for every scenario to imagine you are the person as described and you want to use the Mobina software in the best way possible to improve your processes and IT landscape. You want to make sure that your information is shared with the right people in your company.

There are no strict time limits for the assignments. Some assignments might be less straightforward and therefore take longer, while others might be finished more quickly. You can take some time to get acquainted with the reference model as well. For all the assignments together, a time slot of 1.5 hour is scheduled.

The assignments will be executed on the server we reserved especially for logging. For each of the data collectors a company is created in the software. For all the five assignments you use the same company. We already created some users, applications, and data for these companies. A list of users and applications of the company can be found in Appendix A. Some of the data may be duplicated to some places, but you should not let this influence your behavior. For each assignment you get a different login, since you represent a different person. You will get the login credentials from me when you start an assignment. You can use these logins in a Google Chrome web browser by going to logging.mobina-it.com.

Assignments

In this section, I will shortly introduce the company the people from the different assignment work for. Next, the five different assignments are introduced.

The company

Smart Crane is a company which has about 300 employees and produces hoisting cranes on customer order. About 90% of their orders are cranes which are configured with predefined options. The other 10% of their cranes are specials: the engineering department designs specific solutions for the customer.

The company produces subassemblies based on forecasts of the configuration items and assembles the final product on customer order. The employees are crucial in this process, the crane is a complex product which, for most of the customers, is not engineered in detail but partly assembled and finished off based on the employees' experience.

To achieve short and reliable lead times in combination with a wide range of options for the customers, good information exchange between all parts of the company is essential. The process is complex and delicate, so wrong or missing information can disrupt the whole company. Because of this complexity, Smart Crane is using Mobina to improve interaction and to achieve operational excellence.

Assignment 1: Missing materials

In this scenario you are the production manager. You have been noticing that during the assembly of the product, a lot of your workers are missing the necessary materials. Every time, they have to stop working and go to the warehouse to search for the materials. However, it is almost impossible to find the materials. Therefore, most of the time your workers need to ask the warehouse manager, and often have to wait until he is available. You notice a lot of time is wasted with this and you like to solve this.

Assignment 2: Planning of the warehouse

In this scenario you are the warehouse manager. From experience, you know that it saves a lot of time if the warehouse is well-organized and if all the newly delivered supplies are in the right place. However, currently you have no overview about what supplies are coming in. Because there is no overview, you have no option but putting them in an empty place at the moment of arrival. You would like to have insight in what is coming in, so you can plan your warehouse conveniently and more structured to make the entire process more efficient.

Assignment 3: Overhead engineering new product

In this scenario you are an engineer. For the development of new crane variants, you use the software program CAD. You know that in most of the cases a variant only has one or two differing variables. For example, only the length need to be changed. However, you are not able to use the models of the other variants, since the CAD system doesn't allow you to easily change part of a variant. When you change the length of the crane, you have to move all components on the crane manually to their new position. Therefore, every time you make a new variant, you must engineer the entire crane again, which takes a lot of extra time. You feel like this is a waste of time, especially since this is also not very interesting work to do.

Assignment 4: Lead time reduction specials

In this scenario, you are the COO of Smart Crane. From the latest reports, you have learnt that the specials take too long to produce. This has led to increasing lead times and also large costs, because they take up a lot of time of your employees. The lead times have increased so much, that solving this has very high priority for you and your company. But you can't solve this on your own, so you want to involve your people. You decided to start with the two persons who can probably tell you the most about this, the head of engineering and the production manager.

You want to use Mobina to find out if they already thought about this and found some good solutions. If these solutions are existing, you also want to give them feedback about what you think about these solutions. Besides seeing what is already done, you also want to motivate the head of engineering and the production manager to find all bottlenecks and how these bottlenecks potentially can be solved.

Assignment 5: Compatibility systems

In this scenario, you are a planner. The planning module of the ERP system of Smart Crane is not sufficient. Therefore, you use a separate system, PlanIT. PlanIT loads the customer orders from the ERP system as well as estimated times for production steps. After you fix a planning slot, PlanIT puts the planning data in the ERP system, so it can be used for release of production orders. However, if for example a purchase order or preceding production order gets delayed, you don't get notified. Therefore, you have to manually check the ERP system to ensure that your planning can still be maintained. If a production order is delayed, you have to shift it to a later time slot and potentially move other orders to fill up the time slot. This manual check takes a lot of extra time and also has a high risk of errors. You would really like to find a solution to make the systems interoperable, or at least find a way that you get notified automatically in the case of changes or problems.

Appendix A

Users in your company

- CEO
- CFO
- CIO
- Controller 1
- COO
- Engineer 1
- Engineer 2
- Head of Engineering
- HR Manager
- Planner 1
- Planner 2
- Production Manager
- Production Worker 1
- Production Worker 2
- Purchaser 1
- Quality Manager
- Sales Manager
- Warehouse Manager

Apps in your company

- Asset management
- CAD
- CRM
- ERP
- ERP – Master data
- ERP – Purchasing
- ERP – SCM
- ERP – Service management
- ERP – Shop floor module
- ERP – Warehouse management
- PDM/PLM
- PlanIT
- SCM
- Website and -shop