

UTRECHT UNIVERSITY

MSC. BUSINESS INFORMATICS, MBI

MASTER THESIS

**Flexing the Interface:
The Role of Discourse Knowledge for generating UIs**

Author:

Marc Selles

First Supervisor:

Prof. Lynda Hardman

Second Supervisor:

Dr. Sietse Overbeek



October, 2018

Abstract

Discourse knowledge has been made explicit for identifying the presentation structure of static documents and User Interfaces (UIs) for multimedia presentations. These interfaces, however, differ from the interfaces for functionalities of an application. Our research investigates the role of discourse knowledge for generating UIs for application functionalities. The UIs should adapt to a change of screen size or user requirements. We have developed a framework that uses both domain and discourse knowledge as well as layout information for generating UIs. This framework has been applied to three use cases in the domain of Human Resources. Each use case represents a single functionality of the application. The process of generating the use cases showed three main findings. The presentation structure of the functionality can be adjusted by the use of discourse knowledge. Additionally, meaningful varying layouts of a single functionality can be generated while preserving the presentation structure. Lastly, discourse knowledge can be used to identify additional information from the functionality that can be hidden if necessary.

Acknowledgments

This master thesis represents the final step of my Master of Business Informatics at the University of Utrecht. I would like to express my gratitude to the many people who have supported me on my journey. Initially, I would like to thank my first supervisor, Lynda Hardman, for guiding me through this research. Her expertise of discourse knowledge and User Interfaces pointed me in the right direction and her feedback and our discussions provided me with valuable insight. Thank you for all our sessions from which I have learnt a great deal! Additionally, I would like to thank Sietse Overbeek for being my second supervisor and that you were always accessible for me.

Furthermore, I want to thank Han Gaaikema and Jolien Van Aken for giving me the freedom of doing my research with their company. I enjoyed our sessions a lot! Additionally, I would like to thank my loving girlfriend, Jette, for supporting me and thinking along with me when necessary.

Last but not least, I would like to thank my family for supporting me and always believing in me. This last section has not always been easy for me, but the support of my family and Jette helped me a lot.

Table of content

Abstract	III
Acknowledgments	V
1 Introduction.....	1
2 Related work	3
2.1 <i>Domain model</i>	3
Use of Domain Knowledge for Generating User Interfaces.	4
2.2 <i>Discourse model</i>	4
Use of Discourse Knowledge for Generating User Interfaces.	5
2.3 <i>User Interface Components</i>	6
Use of UI Components for Generating User Interfaces.	6
3 Research Questions.....	7
4 Research Method.....	7
5 Domain Knowledge.....	8
5.1 <i>Modelling the Domain</i>	8
5.2 <i>Domain Model Implementation</i>	10
6 Discourse Knowledge.....	11
6.1 <i>Modelling the discourse</i>	11
Identifying the UI Elements.	11
Grouping the UI Elements	12
Identifying the Relationships between (Groups of) UI elements.	13
6.2 <i>Discourse Model Implementation</i>	16
7 Layout Information	18
7.1 <i>Specifying the Layout</i>	18
Use of UI Components.	18
Adjusting the Layout Due to Screen Size.	20
7.2 <i>Implementation of the UI Components</i>	21
8 Implementation of the Framework	22

9	Evaluating the role of Discourse Knowledge in a framework	24
9.1	<i>Adjusting the Presentation Structure of the UI.....</i>	25
9.2	<i>Generating Multiple Layouts of a Single Functionality.....</i>	26
9.3	<i>Adjusting the Layout Due to a Change in Screen Size.....</i>	28
10	Conclusion & Future Work.....	29
11	References	30
12	Appendix.....	33
	Appendix 1: Functional requirements	33
	Appendix 2: Classes, attributes and relationships for functionalities	35
	Appendix 3: Adjustments of the domain model	37
	Appendix 4: Elements and structure of functionalities.....	39
	Appendix 5: Relationships of the elements of functionalities.....	42
	Appendix 6: RST's of functionalities	44
	Appendix 7: XML of RST's	45
	Appendix 8: Style guide of the use cases.....	47

1 Introduction

Many companies develop applications for use by their customers. For optimal usage, the application should be suitable for multiple users on multiple devices. The user interface (UI) is the part of the application that a user can see and with which s/he can interact. The UI should ensure that the user understands the application and knows how to use it (Galitz, 2007). The development of a UI, however, is a time-consuming and costly process (Calvary & Coutaz, 2014), especially when taking multiple users and devices into account.

Nearly all applications consist of multiple UIs in order to utilize all functionalities of the system. Each of these functionalities should be part of multiple UIs for different devices to exploit the device-specific capabilities (Calvary et al., 2003; Macik, Carny & Slavík, 2014). For example, a mobile phone has a smaller screen than a laptop, so the UI should take this difference into account. In addition, users' requirements may differ (Maguire, 1999; Calvary et al., 2003; Vairamuthu & Anouencia, 2018), which lead to tailored UIs. Developing this wide variety of UIs manually is time-consuming and therefore costly. Generating UIs with the use of explicit knowledge may solve this problem. In order to do this, knowledge should be available to guide this process. To generate multiple suitable UIs, knowledge about the domain of the application is necessary in order to know which information is related to each functionality (Geurts, Bocconi, Van Ossenbruggen, & Hardman, 2003; Visser, 2011; Cranefield, 2001). The next question is how to structure this information in the UI. For example, the title of a functionality is displayed before its content. In conclusion, generating UIs using knowledge about the application domain and structure can solve the problem of developing all UIs manually for functionalities of an application.

Although domain knowledge is required to generate UIs (section 2.1), it is, in itself, insufficient. How the domain knowledge should be structured in the presentation, also known as the presentation structure, is often neglected or remains implicit, whereas it is needed in order to guide potential changes in the UI. This research therefore investigates a way to use this presentation structure. Knowledge about the presentation structure is referred to as discourse knowledge (section 2.2), which clarifies how the information of the functionality - the message that is intended by the communication - will be transferred to the user in the right way (van Ossenbruggen, Geurts, Cornelissen, Hardman & Rutledge, 2001).

Geurts et al. (2003) developed a process where this discourse knowledge is used, in combination with the domain knowledge, to generate multimedia presentations. Similar to the UIs of functionalities of an application, multimedia presentations can consist of text, video, animations and sound. However, the presentation types differ concerning both the elements and the relationships. Some of the UI elements require the input of the user, possibly causing a change in the elements. Multimedia presentations elements do not require such interaction. They can be replaced with others, but the media items themselves do not change. For example, subtitles of a movie are dynamic, as text is replaced continuously. Additionally, the UI typically consists of multiple small elements, containing only a few words or small images. As a result, a UI typically consists

of a greater number of smaller elements than those in multimedia presentations, which mainly consist of one text, made up of multiple sentences plus a few other elements such as images. The last main difference is the purpose. Multimedia presentations have only one purpose, namely to show information about one subject. UIs do not only show this, but they can also gather information about multiple subjects. In conclusion, multimedia presentations and UIs for applications are related to each other without being one and the same thing. The process of generating multimedia presentations can therefore be used as an inspiration to generate flexible UIs.

The information and structure of UIs can be made explicit with the use of domain and discourse knowledge. However, how this information should be shown by the UI also needs to be specified. A style guide can be used to record the UI layout. The characteristics of the elements of the UI, such as buttons and input fields, are defined in the style guide. Additionally, when needed, the change in elements when the screen size changes can be specified in the style guide (Laubheimer, 2016) (section 2.3). As the style guide only records the layout information, it does not include the knowledge about the layout decision. The process of generating flexible UIs with the use of both domain and discourse knowledge as well as layout information is shown in Fig. 1.

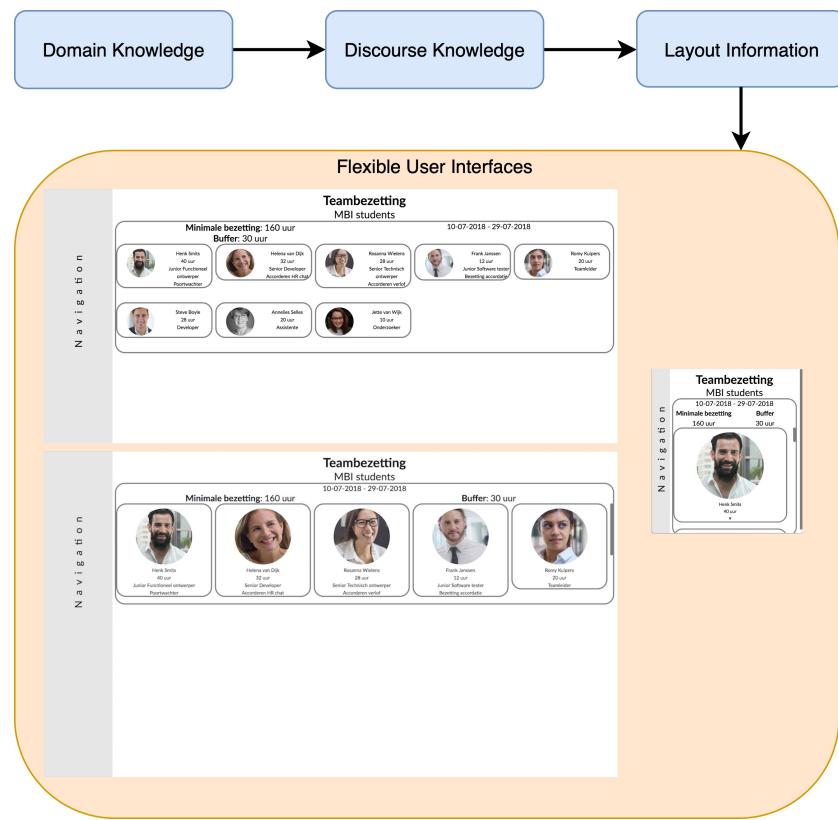


Fig. 1. The Process of the Generation of Flexible User Interfaces for a Functionality of a Use Case (Used Portrait Pictures by Antonio, 2015, Licensed under Place it Blog).

This research focusses on the role of discourse knowledge for the generation of flexible UIs for functionalities of an application. Research questions are set-up in order to investigate this role (section 3). The discourse knowledge structures the elements of the UI. It does not, however, define the layout of these elements. This research does not focus on the development of an aesthetic design but rather on the generation of multiple UIs for different screen sizes and aspect ratios. This is carried out through the use of discourse knowledge incorporated into a framework, developed on the basis of a case study (section 4). For this framework, domain knowledge is modelled (section 5) to identify the information of the application domain. Discourse knowledge is then made explicit to investigate the role of this knowledge in that generation of UIs (section 6). In order to display the information in the right way, the UI layout is specified (section 7). This whole process is implemented (section 8) in order to generate flexible UIs. At the end of the research, the added value of using discourse knowledge will be evaluated (section 9). Based on the findings of this evaluation, a conclusion is drawn, and more research will be proposed (section 10).

2 Related work

In order to generate the UIs which represent a functionality, three types of knowledge or information should be present. To be able to identify the information that will be used, knowledge about the domain and about how this domain can be modelled is explained. After this, discourse knowledge and a way to make this knowledge explicit will be looked at in more detail. Finally, both the purpose and the use of UI components are described.

2.1 Domain model

According to Blok & Cybulski (1998):

Domain models ... help keeping consistency in the design of previously developed and new systems. They allow a more thorough and complete specification of the system functions and data. They facilitate systematic reuse of software across the entire development life cycle, and hence, make the development of systems in a given problem area more effective and efficient. (p. 2).

This domain understanding and description can be obtained by the use of domain analysis methods. Using these methods, solutions to a domain can be identified, organised and modelled so that concepts in the domain can be reused (Ferré & Vegas, 1999).

In our research, all information that is used in the application is stored in the domain model. There is not a single specific source from which this information needs to be captured. However, in this research, the information is obtained from the functional requirements of a system as specified by a customer. The functional requirements describe all the expected functionalities of the application. A domain model can be made using this information.

In order to model a domain, UML notation is often used (Sobernig, Hoisl & Strembeck, 2016; Morisio, Travassos & Stark, 2000; Blok & Cybulki, 1998; Störrle, 2013; Felfernig, Friedrich & Jannach, 2000). UML notation describes an abstract description of system behaviour, the system architecture, and a level of detailed design (Morisio et al., 2000). It is widely adopted as a standard design method in the industrial software development process (Felfernig et al., 2000). Advantages of the UML notation are not only that the UML infrastructure provides built-in implementation techniques for domain-specific modelling language for reusing and extending the UML (Sobernig et al., 2016), but also offers industry-grade tool support for UML (Sobernig et al., 2016, Störrle, 2013). This is comprehensible (Felfernig et al., 2000) and has also been subjected to scientific evaluations of its semantic foundations (Sobernig et al., 2016). Our domain information, including the objects and relationships among them, needs to be modelled to allow connections to be specified with appropriate discourse knowledge. We have chosen to use the UML class diagram, as this notation captures the structure of interacting system objects, their classes and their relationships. According to Fakhroutdinov (n.d.), “a class is a classifier which describes a set of objects that share the same features, constraints and semantics.”. A class usually consists of properties, called attributes. To give a simple example, a class can be “Customers”, which contains the attributes “ID”, “name” and “gender”.

Use of Domain Knowledge for Generating User Interfaces.

This research focusses on the role of using discourse knowledge for generating UIs. In order to make the discourse knowledge explicit, the elements of the UI need to be identified. These elements are identified using the domain model, which is needed to make the information on the domain transparent. Since previous research shows what the role of a domain model is for generating UIs (Gennari et al., 2003; Höök, 2000), we will not elaborate on it.

2.2 Discourse model

Discourse knowledge is seen as the knowledge about how information will be transferred to the user (van Ossenbruggen et al. 2001). This can be done by looking at the form of writing. One can look at how segments in a text should be compared with each other. For example, an antithesis will often be represented with the conjunction “but”. By making this discourse knowledge explicit, the discourse functions of the text can be specified (Scardamalia & Paris, 1985; Carlson, Marcu & Okurowski, 2003; Taboada & Mann, 2006; Mann & Thompson, 1987; Azar, 1999). The segments of a text are often related to each other, for example a segment of text can be an antithesis to a segment with a statement. These relations can be described by concepts of discourse knowledge.

The discourse knowledge specifies the functions of the segments of a text and the relations among the segments. This makes it clear which segments belong together. Coherent information in the text can be accumulated, whereas unrelated information can be separated. Knowledge of this structure can then be used to determine the layout. The discourse knowledge can therefore be seen as the presentation structure. Making

the discourse knowledge of text explicit, Bateman, Kleinz, Kamps & Reichenberger (2001) produced two document layouts with minimal and no layout decisions. Coherent segments of text were put together and separated from the non-coherent segments. Unlike document layouts, the layout of UIs must be flexible, as it should adapt to different screen sizes and requirements by users. Our research, therefore, investigates how discourse knowledge can be used to determine potential acceptable layouts of UIs.

To make discourse knowledge explicit, Bateman et al. (2001) make use of Rhetorical Structure Theory (RST). RST is developed to describe the structure of text with the rhetorical relationships between segments of the text. Originally, 23 relationships were defined by Thompson & Mann (1987). A few relations have been added since.

RST relationships can be either asymmetric or symmetric. An asymmetric relationship, also called a multinuclear relationship, consists of two segments of text. These two segments are the nucleus and the satellite. The difference between the two segments is that the nucleus segment is seen as being more essential to the writer's purposes than the satellite segment. A symmetric relationship, also called a presentational relationship, contains two or more segments. These segments do not have this degree of importance (Batemen et al., 2001).

For example, statement [1] in Fig. 2 gives basic information about a topic, while [2] and [3] give additional information to back it up [1]. This asymmetric relationship (a) leads to [1] as the nucleus, as it contains more essential information than [2] and [3], the satellites. In this case, [2] and [3] elaborate on the main statement [1], resulting in the relationship *elaboration*. For the relationship between [2] and [3], a multinuclear relationship (b) is used, as both elements are equally important; [2] is neither more or less important in the text than [3]. Additionally, it does not matter whether [2] or [3] is mentioned first. Therefore, the relationship *joint* is identified, which is a presentational relationship where order does not matter.

[1] History students generally find it difficult to find a job after their study. [2] This is mainly due to both the large numbers of graduates [3] and the amount of employment in this field.

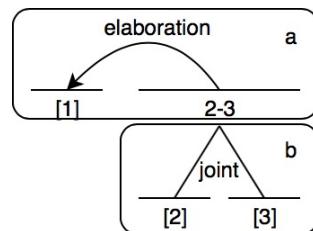


Fig. 2. Examples of RST Relationships: *Elaboration* and *Joint*

Use of Discourse Knowledge for Generating User Interfaces.

Discourse knowledge has been used in previous research for analysing the structure of text. By making this knowledge explicit using RST, flexible layouts can be generated, as demonstrated by Bateman et al. (2001). The RST makes the structure of the text explicit by identifying the relationships between separate parts of the text (André & Rist, 1993). In our research, we share the same purpose, namely to structure the UI using the relationships among its elements. Although RST has so far only been applied

to static documents using both text and images, we will use RST to make the discourse knowledge of UIs explicit.

Additionally, Geurts et al. (2003) developed a process whereby domain, discourse and layout knowledge have all been applied in order to generate multimedia presentations. Discourse knowledge has been used in order to structure the knowledge of the domain. This was done by mapping the domain knowledge. In the end, design knowledge was applied in order to ensure that the elements of the multimedia presentation were displayed correctly. This process of firstly identifying the information that should be present, then structuring this information and at the end displaying it is similar to the process of generating UIs. The research by Geurts et al. can therefore be used as an inspiration for our process. However, it focusses on generating multimedia presentations using only a single discourse structure. When generating UIs of an application, multiple discourse structures need to be developed to realise all the functionalities of the application. Therefore, the information needed at different stages in the process can differ.

In conclusion, the process used by Geurts et al. (2003) to generate multimedia presentations can be used to as an inspiration. We have used and adjusted RST in our research in order to structure the information of the domain and make the discourse knowledge explicit.

2.3 User Interface Components

The discourse knowledge structures the information that will be presented in the UI. However, since it does not contain layout information, it does not state how the information should be presented. For example, an image is a different way of presenting than a text. The decisions that should be made for presentation are defined in the style guide. In the style guide, we created templates to specify the properties of the presentation. These templates are called UI components and indicate what kind of UI object is used (e.g. image or text). The elements of the RST are mapped to appropriate UI components and the properties of the UI components are filled in appropriately. For example, the property that specifies the font-size of a UI component representing a text with standard font-size should be about 12px instead of 100px. An element can be represented in many ways. Both the mapping and filling-in of the properties are therefore done manually.

Use of UI Components for Generating User Interfaces.

Since discourse knowledge does not specify the layout of the elements in the UI, some means of doing this is needed. We introduce the UI component to do this. Changes in the UI layout, due to a change in screen size or in user requirement, are specified using UI components.

Mulder (2016) developed a language to define UI components by their fundamental properties. The aim of the language is to improve the process between a design and the corresponding code, so that the properties of this language describe a component in detail. Properties, such as width, height, content, and also the exact positioning on the

screen and animations of the UI component are defined. In our research, we will use this way of specifying the UI component. The UI components will be specified by properties in order to determine the layout of the UI elements. Our research, however, does not focus on the development of an aesthetic design. The UI components are therefore not specified at the same level of detail as Mulder presented them. Additionally, Mulder does not take into account the change of the elements of the UI due to a change in screen size. One way to do this is by using UI components. How the layout of a UI can be adjusted due to a change in screen size will be investigated in our research.

3 Research Questions

Our main research question is as follows:

What is the role of discourse knowledge for generating flexible UIs for an application's functionality?

We focus on UIs that take either a change in screen size or in user requirement into account. A variety of user requirements will result in multiple layouts that have to be generated, and a generated UI can be adjusted if a screen size is changed. For the latter, the transition of the layouts should be done smoothly. In this way, the overall layout will be preserved so that the user will not be bothered by any changes. This means, however, that the process of applying the UI components, which are used for the layout information, could differ from generating multiple layouts due to the variety of user requirements. The role of discourse knowledge will be evaluated using only the following sub-questions:

- How can the presentation structure of a single functionality be adjusted?
- How can multiple layouts be generated while preserving the presentation structure and screen size?
- How can a generated layout be adjusted if a screen size is changed while preserving the presentation structure?

4 Research Method

A framework was developed to investigate the research questions. The framework was developed during the realisation of the development of an application for a company in the domain of Human Resources (HR). Using the framework, three functionalities - referred to as use cases - were developed to investigate the research questions. The development of one of these use cases has been elaborated on in this paper. The necessary steps to be taken in order to develop all use cases can be found in the appendix indicated.

The process for the generation of the UI that is represented in Fig. 1 has been used for the development of the framework. Firstly, domain knowledge of the application is made explicit in a domain model. The discourse knowledge of the use cases is then

modelled using RST. UI components are developed and matched with the elements of the RST in order to represent these RST elements in the UI. The generated UIs of the use cases are used to answer the research questions.

5 Domain Knowledge

5.1 Modelling the Domain

All functionalities of the application are described in a functional requirements document supplied by the company. These requirements should contain the information that should be presented in the UI. As stated in section 4, one use case, with the following functional requirements¹, will be used as a running example:

View team requirements

A team, consisting of any number of employees, varying from 0 up to as many as 40, must indicate how many working hours are – probably – needed for each period in the year to complete the work that needs to be done. This is represented as the minimum person hours of the team. A buffer is used to take into account any possible dropout, due to illness, for example.

In order to achieve the minimum person hours plus buffer, the working hours of all team employees in the period should be explicit. Additional information about the function and role of each employee indicates what the employee will be doing in the period.

The functional requirements of all use cases can be found in Appendix 1.

The domain information captured by the functional requirements is modelled using UML. Classes and attributes are identified to provide an overview of the information used in each functionality. These classes and attributes can also be reused for other functionalities. Normally, operators are identified in UML. This research, however, focusses on the UIs, and not the back-end of the application. The information of the operators does not necessarily have to be used and is therefore omitted.

After identifying the classes and attributes, relationships are established between the classes. All classes, attributes and relationships of the use case are shown in Table 1. All information for the use case is contained in the functional requirements, with the exception of the role types. Five role types, such as “Poortwachter” and “Bezetting accordatie”, are identified in the functional requirements in another functionality. As the role type is included in the use case, these identified role types are reused. The classes, attributes and relationships of all use cases can be found in Appendix 2.

¹ The functional requirement is originally written in Dutch and translated to English.

VIEW TEAM REQUIREMENTS	
Class	Attribute
Team	naam
Teambezetting	minimaleBezetting [label;value;units], buffer [label;value;units], datumBegin, datumEinde
Medewerker	voornaam, achternaam, foto
Functie	naam
Rol	naam
Bezetting Accordatie	
Poortwachter	
Verlof Management	
Responsetijd Management	
HR Chat Management	
Relationships	
[Medewerker - Functie 0..1] [Medewerker - Rol 0..*]	
[Rol - Bezetting accordatie 0..1] [Rol - Poortwachter 0..1] [Rol - Verlof Management 0..1] [Rol - Responsetijd Management 0..1] [Rol - HR Chat Management 0..1]	
[Team - Medewerker 0...40] [Team - Teambezetting 1]	

Table 1. Classes, Attributes and Relationships of the Functionality

The information required for the functionalities was validated by means of four sessions with two experts in the HR domain. The most important adjustments that were made as a result of the sessions with experts can be found in Appendix 3 “Adjustments of the domain model”.

A domain model containing the classes, attributes and relationships of all functionalities of the application has been constructed, see Fig. 3, as the whole application consists of more functionalities than simply those in the three use cases. The semantics of the domain model have, however, not been perfected, but are sufficient to make the information of the functionalities explicit.

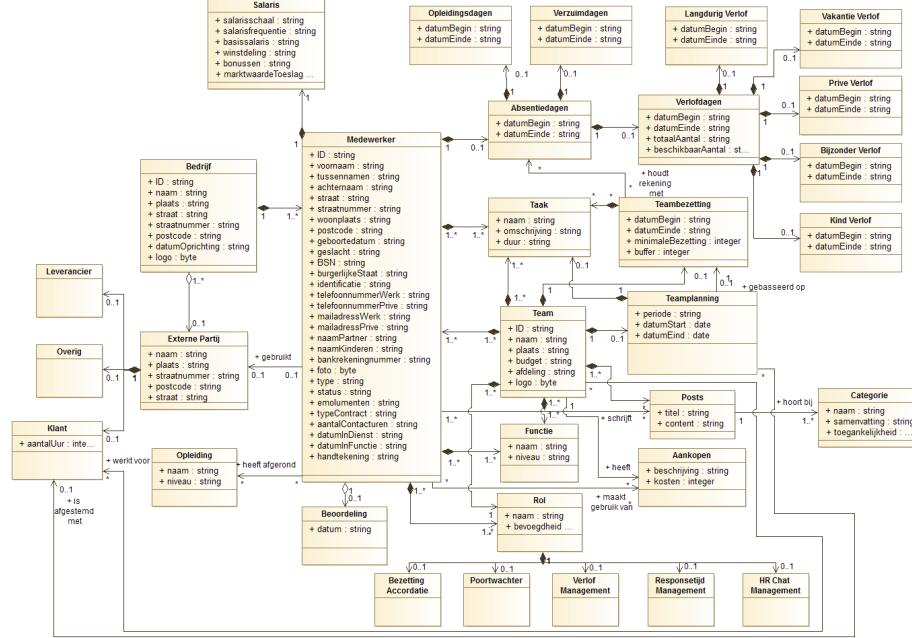


Fig. 3. HR Application Domain Model

5.2 Domain Model Implementation

The domain knowledge is modelled using the open source modelling environment “Modelio” (Version 3.7.1, 2018). The domain model was exported to the XMI format, a format that is used to format metadata in XML, in order to put the framework into practice. The data, containing information on the functionality that is to be developed, is delivered in a CSV file. This data is entered into the XMI model using an XLST transformation, as shown in Fig 4.

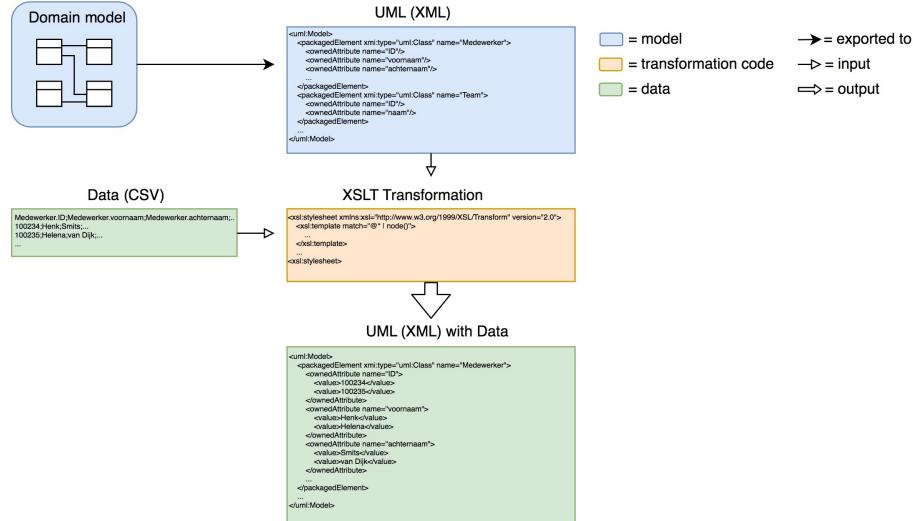


Fig. 4. Process of Filling the Domain Model with Functionality Data

6 Discourse Knowledge

6.1 Modelling the discourse

The discourse knowledge should be used in order to identify the presentation structure of the UI to guide its elements. In order to model this discourse knowledge, three steps using RST are taken. RST makes the relationships between segments of text explicit. A UI, however, does not only consist of text. Instead of the relationships of segments of text, the relationships of the elements of the UI should be identified, such as images, text fragment and input fields. Therefore, the elements of the UI should be identified first. These elements do not, however, contain layout information. Secondly, the related UI elements should be identified and grouped, as is done by text in Fig. 2. Finally, the relationships amongst the UI elements and groups are identified.

Identifying the UI Elements.

The domain model captured the information that should be present in order to use the functionalities of the use cases. This information in itself, however, is not always sufficient to represent an element in the UI. For example, in our use case the element in the UI representing the period consists of two attributes of the domain model: both the start and end dates, separated by a line. Additionally, some elements of the UI do not contain the information of the domain model, such as buttons. The UI elements should therefore be defined in order to represent the functionality, which is shown in Table 2.

VIEW TEAM REQUIREMENTS	
UI Elements	
1	Teambezetting
2	{Team.naam}
3	{Teambezetting.datumBegin} - {Teambezetting.datumEinde}
4	Minimale Bezetting: {Teambezetting.minimaleBezetting} uur
5	Buffer: {Teambezetting.buffer} uur
6	{Medewerker.foto}
7	{Medewerker.voorNaam} {Medewerker.achterNaam}
8	{Medewerker.aantalContacturen} uur
9	{Functie.niveau} {Functie.naam}
10	{Rol.naam}

Table 2. UI Elements of the Use Case “View Team Requirement”

Grouping the UI Elements

The identified elements of the UI must then be structured in order to make the discourse knowledge of functionalities explicit. Structuring the UI elements starts by identifying which elements do and do not belong together. These elements will be grouped and therefore distinguished from the other elements. The groups can not only contain individual UI elements, but also other groups. The identified groups of the use case under discussion are shown in table 3. Appendix 4 shows the identified elements and groups for all use cases.

VIEW TEAM REQUIREMENTS	
Groups	
1 - 2	Element 1 is the title and element 2 the subtitle. These two elements can therefore be grouped together to distinguish the (sub)titles from the content.
3 - 10	All elements except the (sub)titles are the actual elements needed to perform the functionality.
4 - 5	Elements 4 & 5 both give information about the requirements to be met and are therefore grouped.
6 - 10	Elements 6 up to and including 10 are all information about an employee.
9 - 10	Elements 9 & 10 both contain extra information about the employee.

```

graph TD
    A([1-2])
    B([3-10])
    C([1, 2, 3, 4-5])
    D([6-10])
    E([4, 5, 6, 7, 8])
    F([9, 10])
    A --- B
    C --- D
    E --- F
  
```

Table 3. Structure of the UI Elements of the Use Case “View Team Requirement” (shown in Table 2)

Identifying the Relationships between (Groups of) UI elements.

Relationships of the elements and groups must be identified in order to complete the structure of the UI elements. All relationships needed for our use cases are present in the RST of static texts.

The RSTs of the three use cases require four types of relationships (sequence, joint, volitional result and elaboration), as shown in Table 4. The relationships between the elements of the use case under discussion, are explained in Table 5. The relationships and explanation of all three use cases are given in Appendix 5.

M multinuclear Relationships		
Title	Description	Occurrence in use case*
Sequence	<i>Sequence</i> determines the order in the presentation in which the elements can be seen. In the RST, the elements are placed from left to right. The element on the left will therefore be placed first on the screen. The UI components determine whether the elements are placed from top to bottom or from left to right.	1, 2, 3
Joint	The elements of the relationship called <i>joint</i> occur in random order. The UI components determine whether the elements are placed randomly from top to bottom or from left to right.	1
P presentational Relationships		
Title	Description	Occurrence in use case*
Volitional Result	One element (Nucleus) is the element with which the user makes a conscious interaction with the UI. The other element (Satellite) is the subsequent result of this interaction.	2
Elaboration	One element (Nucleus) contains basic information, which is supplemented with an element (Satellite) containing additional information. The basic information always has priority on the screen in comparison to the additional information.	1, 3

Table 4. Types of Relationships of the RSTs Modelled

VIEW TEAM REQUIREMENTS			
Multinuclear Relationships			
Elements	Relationship	Explanation	
1-2, 3-10	Sequence	The title and subtitle must always be placed before the content on the screen.	
1, 2	Sequence	The title must be placed before the subtitle.	
3, 4-5, 6-10	Sequence	The period determines which requirements are selected and fulfilled. The period must therefore be placed first. Hereafter, the information about the requirements, which are elements 4 & 5, follow. The actual implementation necessary to meet these requirements comes at the end.	
4, 5	Joint	The two elements of the requirements do not have to be a sequel to each other. It does not matter, therefore, which information is put first.	
6, 7, 8	Sequence	A standard format of personal data is used, where the photo of the employee is shown first, then the first and last name and at the end the other information, in this case the number of hours that the employee will work.	
9, 10	Sequence	The function of the employee must be placed before the employee's role.	
Presentational Relationship			
Nucleus	Satellite	Relationship	Explanation
6-8	9-10	Elaboration	As stated in the functional requirements, the function and role of the employee are seen as additional information. As a result, these elements are the satellite elements, whereas the nucleus contains the basic information.

Table 5. Relationships of the Elements of the Use Case “View Team Requirement” (shown in Table 2)

It is possible that elements, or groups of elements, occur several times in the UI, resulting in possible varying completion. However, this must not be visible in the RST, as the elements of the RST should not be duplicated. As a solution to this problem, multiplicity is added to the RST. The multiplicity indicates how often an element, or a

group of elements, may occur in the UI. For example, multiplicity has been used in our use case to represent each employee on the team. The functional requirements of the use case made it clear that a team always consists of a minimum of 0 and a maximum of 40 employees. Multiplicity allows that the employee's information, which is represented in the group containing UI element 6 – 10, can occur 0 to 40 times.

The structure of the use case is specified by identifying and grouping UI elements and by identifying the relationships and potential multiplicity. This structure is used to make the discourse knowledge of the use case explicit using RST, shown in Fig. 5. The RST structure of all use cases can be found in Appendix 6.

The RST structure contains the knowledge about how the elements are related to each other and in which order they should be presented. It does not say anything about either the layout of these elements or their restrictions. The discourse structure therefore represents the presentation structure of the functionality allowing different layouts for its elements.

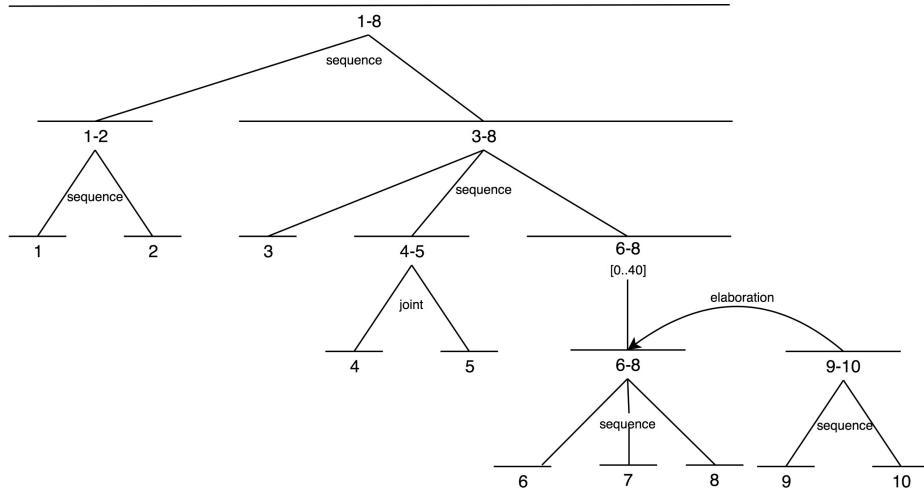


Fig. 5. RST of the Use Case “View Team Requirement” (shown in Table 2, 3 and 5)

6.2 Discourse Model Implementation

The RST is represented in XML for the implementation of the discourse knowledge. This XML format is based on the XML obtained by exporting RST with the open source tool for discourse analysis *rstWeb* (Version 2.0.0; Zeldes, 2017). The RST of Fig. 5 is converted into XML in Fig. 6. The XML of the RSTs of all use cases can be found in appendix 7.

```

<rst>
  <header>
    <relations>
      <rel name="elaboration" type="rst"/>
      <rel name="joint" type="multinuc"/>
      <rel name="sequence" type="multinuc"/>
    </relations>
  </header>
  <body>
    <group id="11"/>
    <group id="12" parent="11" relname="sequence"/>
    <segment id="1" parent="12" relname="sequence">Teambezetting</segment>
    <segment id="2" parent="12">(Team.naam)</segment>
    <group id="13" parent="11" relname="sequence"/>
    <segment id="3" parent="13" relname="sequence">(Teambezetting.datumBegin) - {Teambezetting.da-
turnEinde}</segment>
    <group id="14" parent="13" relname="sequence"/>
    <segment id="4" parent="14" relname="joint">Minimale Bezetting:{Teambezetting.minimaleBezetting} uur</segment>
    <segment id="5" parent="14" relname="joint">Buffer: {Teambezetting.buffer} uur</segment>
    <group id="15" parent="13" relname="sequence" minEntities="1" maxEntities="40" multiplicityId="1" />
    <segment id="6" parent="15" relname="sequence">Medewerker.foto</segment>
    <segment id="7" parent="15" relname="sequence">(Medewerker.naam)</segment>
    <segment id="8" parent="15" relname="sequence">{Medewerker.aantalContacturen} uur</segment>
    <group id="16" parent="13" relname="elaboration"/>
    <segment id="9" parent="16" relname="sequence">{Functie.niveau} {Functie.naam}</segment>
    <segment id="10" parent="16" relname="sequence">(Rol.naam)</segment>
  </body>
</rst>

```

Fig. 6. RST of the Use Case Expressed in XML

The XML of the RST, however, has no knowledge of the meaning of the structure. For example, the XML does not know that elements with the relationship sequence are in a specific order or that the nucleus element is more important than the satellite element when using the relationship elaboration. This knowledge is applied using an XSLT transformation. An XSLT transformation takes the XML as input and applies the knowledge to develop the structure. This process is shown in Fig. 7.

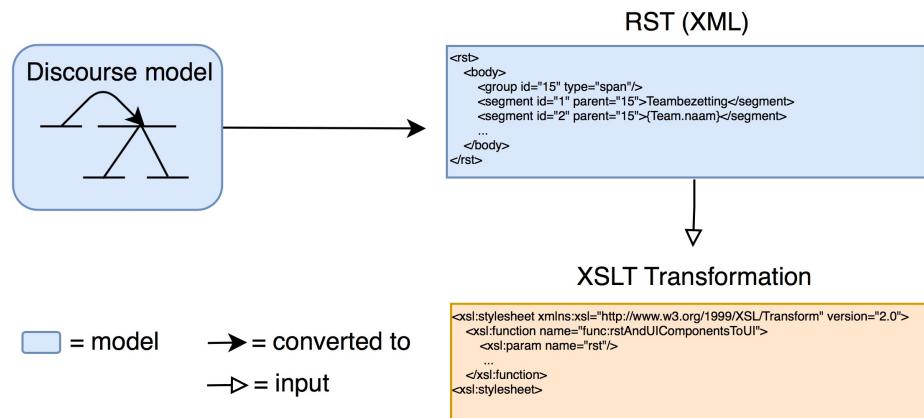


Fig. 7. Process of Implementing the Discourse Knowledge

The discourse knowledge contains both the structure of the use case which includes the UI elements and the relationships between them. It is used to order and prioritise the elements of the UI. The purpose of this structure is evaluated in section 9.

7 Layout Information

7.1 Specifying the Layout

Discourse knowledge is used to structure the elements of the UI. It does not, however, specify how the elements should be displayed in the UI. UI components are defined in a style guide to tackle this problem. The style guide, containing the UI components of all three use cases, is provided in Appendix 8.

Use of UI Components.

UI components record the layout information of the discourse knowledge that can be applied to the UI elements, which potentially can be displayed in several ways. For example, Fig. 8 illustrates that UI element 4 of the use case can be represented as a flowing text, but also with a format containing only the label, value and unit. The most suitable UI component should therefore be selected for the UI element. The shading in Fig. 8 shows what will be shown to the user in the UI.

UI Element: 4
Minimale Bezetting: {Teambezetting.minimaleBezetting} uur

UI Component: Text
De minimale bezetting van het team bedraagt 160 uur.

UI Component: LabelValueUnitInline
Minimale bezetting: 160 uur

Fig. 8. Example of Two Representations of the Same Information

UI components are developed as templates where properties can be filled in. These properties specify the layout and content of the UI component. Two types of UI components have been specified: atomic and composite. A UI component representing a single UI element is an atomic UI component. For example, Table 6 represents the properties of the atomic UI component for representing information as a label, value and unit.

Variable	Font-size	
Variable	Min. font-size	
Variable	Content	Label
Constant		:
Variable		Value
Variable		Unit

Table 6. Properties of the Atomic UI Component “LabelValueUnitInline”

The layout of the groups of UI elements, identified in section 6, is specified by composite UI components. As groups can consist of UI elements and other groups, the composite UI components can contain atomic and other composite UI components. Both these UI components are referred to as the children of the composite UI component. The properties of composite UI components determine not only whether the group will get a border, but additionally whether the children in the group are displayed horizontally or vertically. The order in which the children occur is specified by the discourse model, see section 6. Other properties of composite UI components, for example for specifying a change due to time or interaction, were not required in our research.

To summarise, UI elements and groups are represented in the UI using appropriate UI components. This is done by filling-in the properties of the UI component. An example of this process is given in Fig. 9 as illustration.

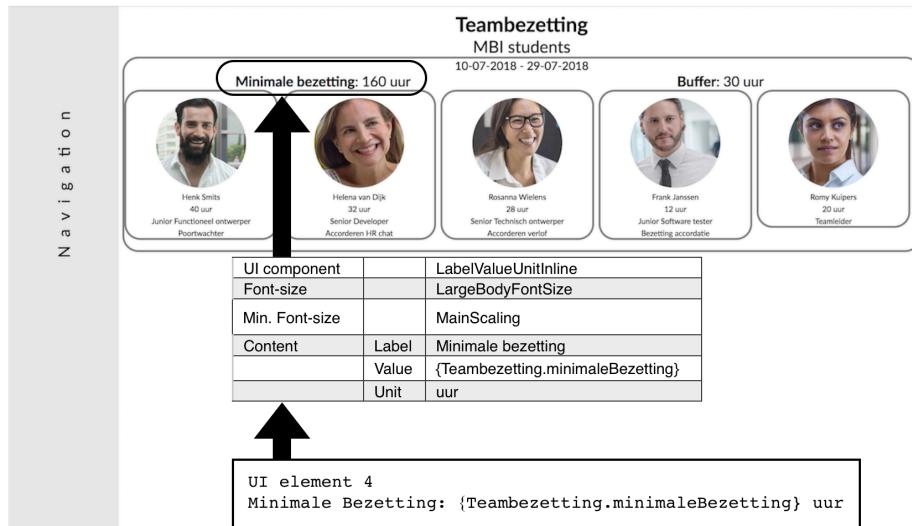


Fig. 9. Representation of a UI Element Using a UI Component (Used Portrait Pictures by Antonio, 2015, Licensed under Place it Blog)

Adjusting the Layout Due to Screen Size.

With a change of screen size, UI components - representing UI element(s) - may no longer fit on the screen. Three steps are taken in this case. For the first step, properties of the UI component can be used to make the UI component smaller. The font-size of a UI component containing text will be gradually adjusted so that the component still fits on the screen until the font-size reaches a minimum. This minimum is specified by the “min. font-size” property. This value is given as a percentage. The minimum font-size is therefore that percentage of the value of the property “font-size”. For example, the values of the properties “font-size” and “min. font-size” are 20px and 60%. If the UI component no longer fits on the screen, the font-size must become smaller to make it fit. This must be done repeatedly until the font-size reaches 12px (60% of 20px). The same process take place for UI components containing images or buttons. The property “min. ratio” is used to specify the minimum value for the properties of “width” and “height” in the same way as “min. font-size” does for “font-size”.

If a UI component still does not fit on the screen, it can be transformed into another type of UI component that does fit. For this process, a custom UI component should be specified besides the original UI component. This custom UI component represents the same UI element, but uses a smaller representation that does fit the screen. To illustrate this transformation, the UI element of Fig. 7 is represented using both UI component “text” and “labelValueUnitInline”. UI component “text” takes more space on the screen than the UI component “labelValueUnitInline”. The original UI component “text” can therefore be transformed into the custom UI component “labelValueUnit” if it does not fit on the screen. Some UI component, however, cannot be transformed into smaller ones. Custom UI components are therefore only specified when applicable.

The last step to be taken is that of hiding UI components. If the custom UI components, or failing that the original UI components, no longer fit onto the screen, one or more of the UI components will not be visible. This, however, does not necessarily mean that the UI components that do not fit will automatically be hidden. The discourse knowledge is used to indicate which element of the group that does not fit on the screen is least important. More space will become available by hiding the UI component of this element. In order to make the elements visible again, we used an arrow for the user to click on. For example, Fig. 10a shows the UI components of the group 6 – 10 of the use case. If the UI components in this group no longer fit on the screen due to a change in the height of the screen, the RST shown in Fig 5. will be used to identify which element or elements should be hidden. Using the RST, the relationship elaboration between the group 6 – 8 and 9 – 10 becomes clear. This relationship elaboration shows that the elements 9 and 10, the function and role of the employee, are additional information. As these elements are both equally important to each other, both should be hidden in the UI, as shown in Fig. 10b.

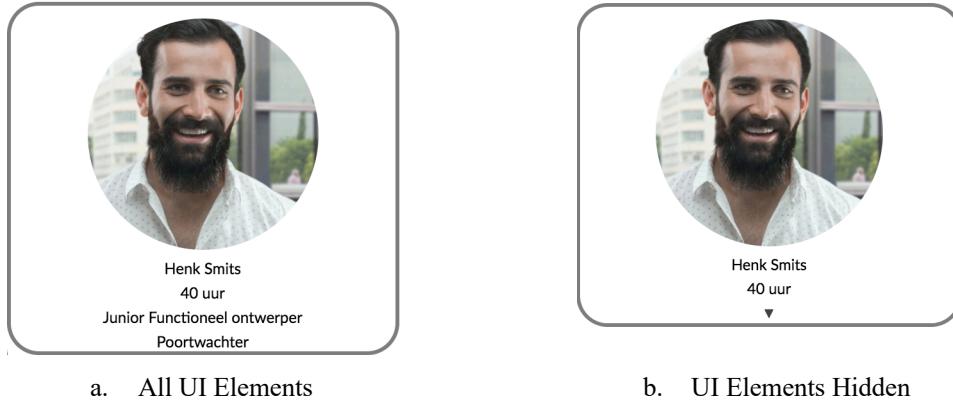


Fig. 10. UI Components of Employee Information (Used Portrait Picture by Antonio, 2015)

7.2 Implementation of the UI Components

The UI components represent the layout of the UI elements. As stated before, the UI components are developed as templates which can be filled with properties. An example of these templates is given in Table 6 and all templates can be found in section on “UI component templates” in Appendix 8. The mapping of the UI element to an appropriate UI component and the filling in of the properties of this UI component is done by an XSLT transformation. This transformation converts the UI component to HTML. The styling of this HTML is developed in CSS.

Javascript is used in order to adjust the UI component if it does not fit on the screen. The steps of changing font-size or ratio, transforming original UI components into custom UI components and hiding UI elements, discussed in section 7, will be applied using this programming language.

The process of the implementation of the UI components is shown in Fig. 11.

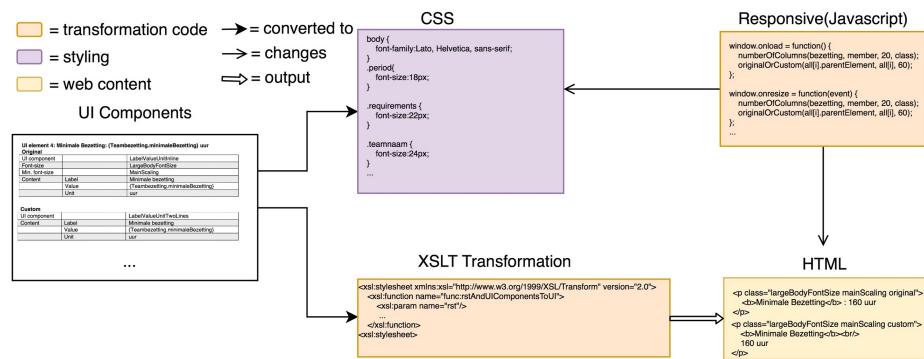


Fig. 11. Process of Implementing the UI Components

8 Implementation of the Framework

The framework consists of both the domain and discourse knowledge as well as layout information. These types of knowledge and information should be combined for the implementation of the framework. To summarize this process, the framework starts by entering data into the domain model (see section 5.1). The RST is then represented in XML and an XSLT transformation is used to apply the knowledge of the discourse (see section 6.1). The UI components are developed into HTML using an XSLT transformation and CSS. Additionally, Javascript is used to handle a possible change of screen size (see section 7.1).

The discourse knowledge, expressed in RST, should be used in the process of the implementation to structure the UI elements into the right position on the UI. The data of the domain model should be used as input for this process, as the UI elements represent the information of the application domain. The UI elements are then transformed into UI components and the properties are filled in. For the whole process, XSLT transformations are used to produce the HTML of the UI. Fig. 12a represents this proposed process.

Unfortunately, the actual implementation of the framework deviates slightly from this process due to technical restrictions. Although the discourse knowledge is still used to structure the UI elements, the process to realise this differs. In the desired framework, the UI elements should be identified using the RST and filled in using the domain model. These UI elements are transformed into UI components at the end for the representation. The UI elements of the actual process, however, are entered manually into a XSLT transformation and are not given by the RST. The discourse knowledge then maps the right UI element, represented using UI components, to the right position in the RST in order to structure the UI. As the UI elements are identified using the discourse knowledge, the actual implementation process, represented in Fig. 12b, should be adjusted for further research.

The actual implementation of the whole framework is represented in Fig. 13. The code of this implementation can be found at <https://github.com/MarcSelles/UI-Generation-Framework>.

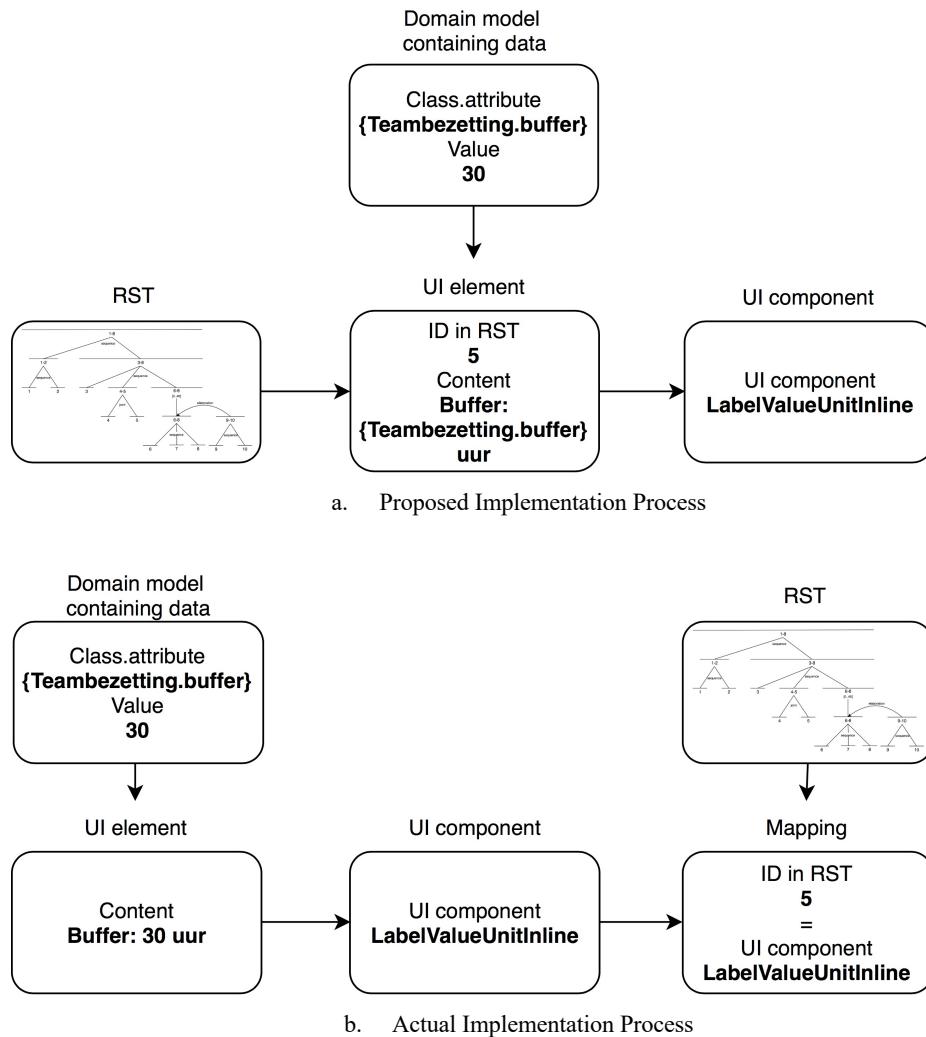


Fig. 12. Proposed and Actual Implementation Process of the UI Elements and Components

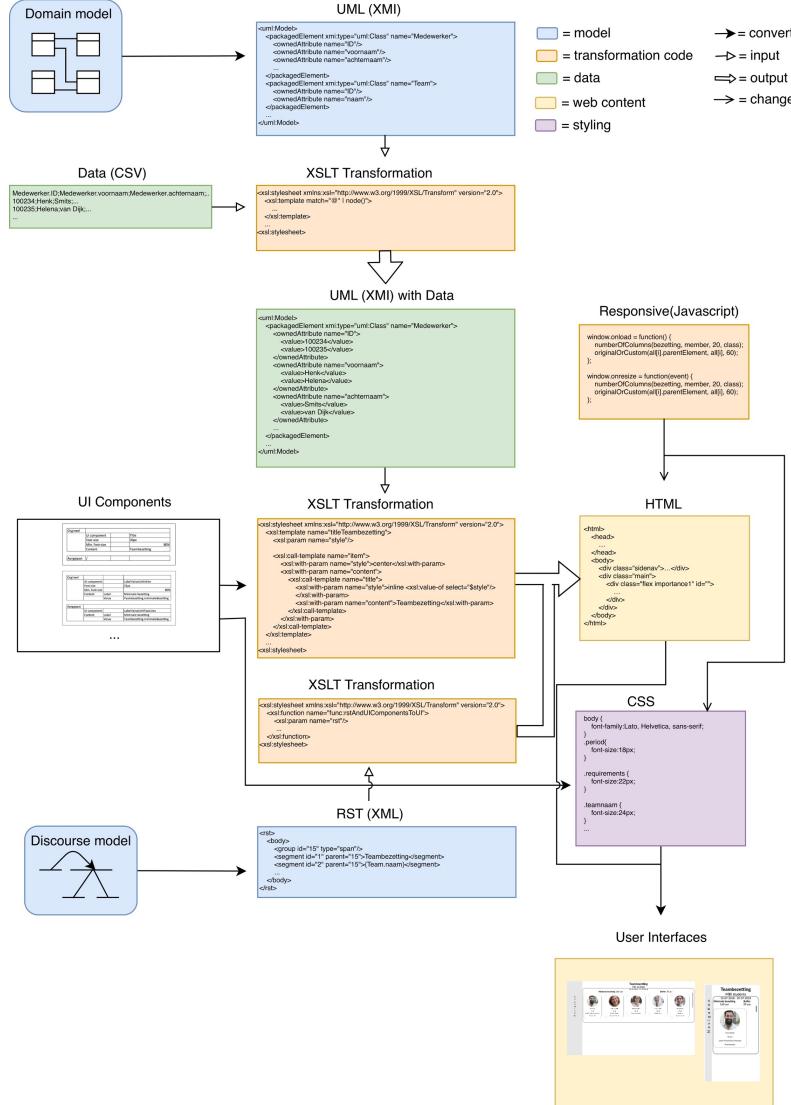


Fig. 13. Actual Process of the Implementation of the Framework

9 Evaluating the role of Discourse Knowledge in a framework

We developed a framework to generate UIs for functionalities of an application. This framework uses discourse knowledge, which had not explicitly been used before for this kind of generation. The discourse knowledge represents the structure of the communication, including the various relationships that are present in it. This is developed

in such a way that the user understands the message that is intended in the communication.

A UI consists of UI elements, which are structured in such a way as to determine the presentation of the UI. A UI therefore contains implicit discourse knowledge. By making this knowledge explicit, it becomes clear which elements do and do not belong together in the UI and also what the relationships between these elements are.

In order to answer the main research question *What is the role of discourse knowledge for generating flexible UIs for an application's functionality?*, the use of the discourse knowledge for the use cases will be evaluated. This will include structuring UIs, developing varying layouts and also changing the layout of the UI resulting from a change in screen size.

9.1 Adjusting the Presentation Structure of the UI

The structure of the UI has been made explicit using RST. This has previously been used in research to make the structure of static documents explicit. By adding the option of using multiplicity, the RST can be used to model the discourse knowledge as well. Adjustments to the structure of the UI can therefore easily be made by adjusting the RST. Fig. 14a shows the generated UI of the use case, where the UI element containing information about the period is given prior to the information about the requirements. Suppose a user wants this to be done the other way round, then the RST would only have to be adjusted, as shown in Fig. 14b. By using the discourse knowledge that is made explicit, the structure of the UI will be flexible.

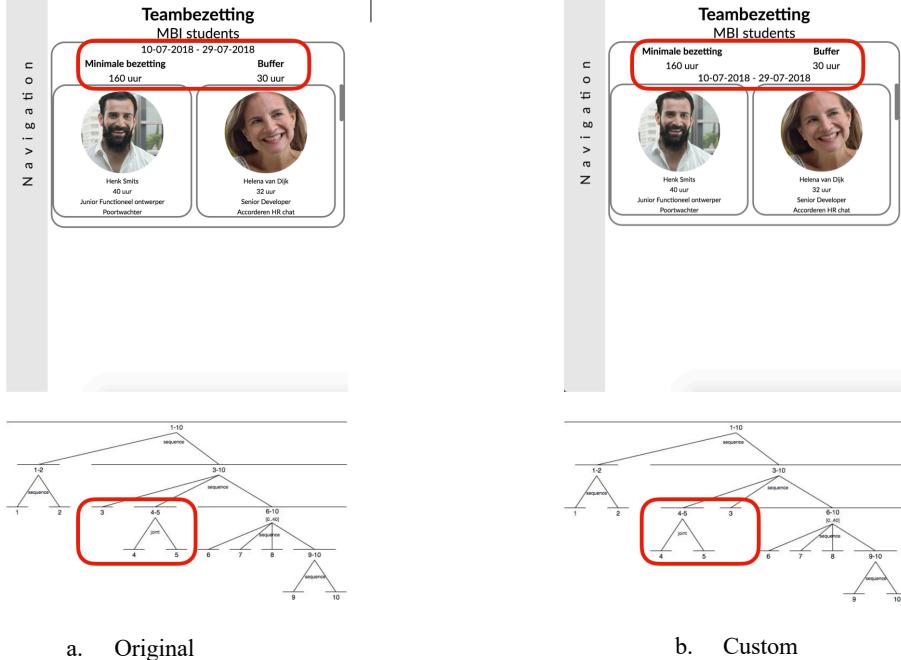


Fig. 14. Generated UIs of the Use Case with Different Structures (Used Portrait Pictures by Antonio, 2015, Licensed under Place it Blog)

9.2 Generating Multiple Layouts of a Single Functionality

The discourse knowledge contains no information about how the element of the UI should be displayed. It only defines the structure of the information to be displayed in the UI. The UI components can therefore be used in multiple ways to provide different layouts with the same structure. Fig. 15 shows that multiple meaningful layouts can be generated of one of the use cases developed for this research, using the same RST. The layout of the elements can therefore easily be adjusted without losing the structure of the UI. This flexibility of layout ensures that the varying requirements of the users can be met.

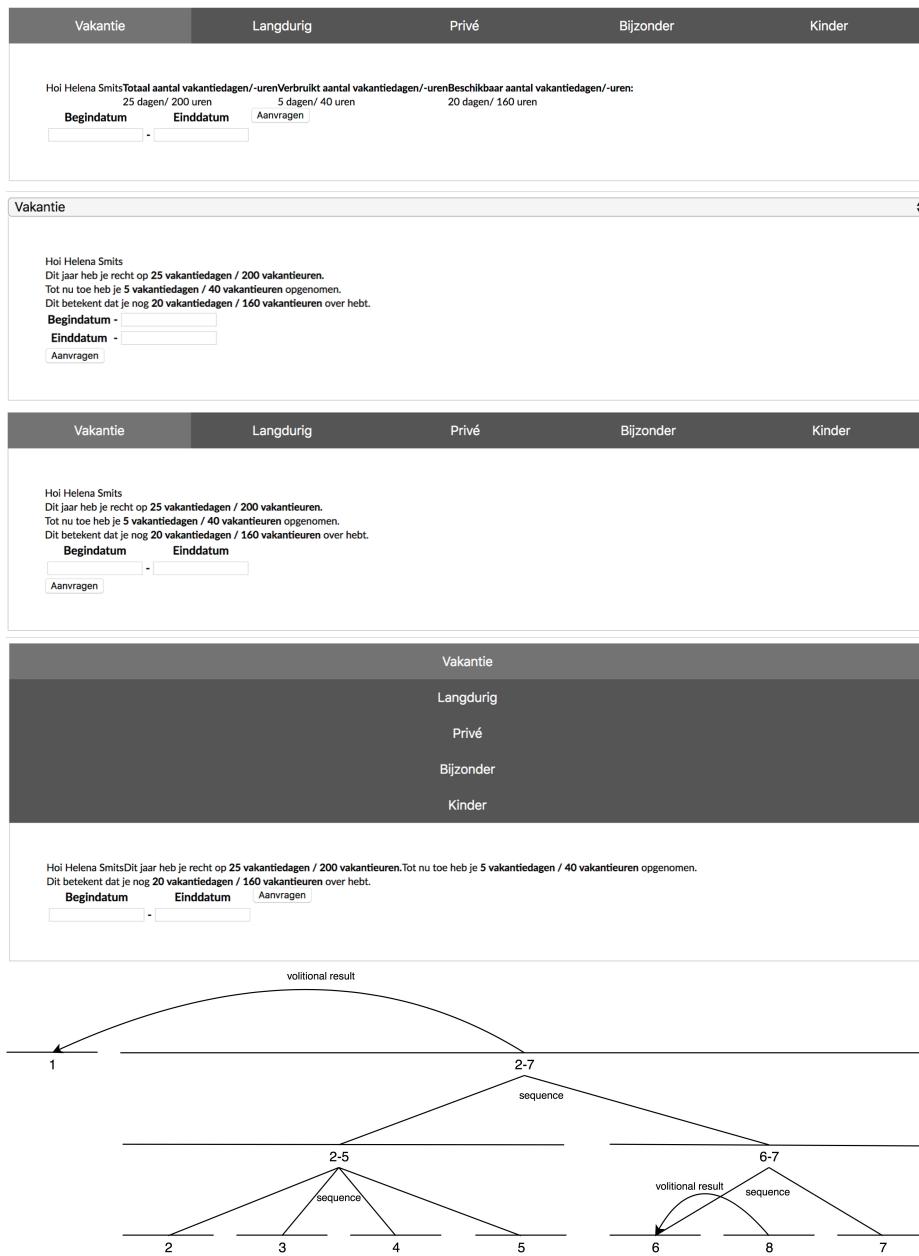


Fig. 15. Generation of Multiple Random Layouts of a Use Case with the Same RST

9.3 Adjusting the Layout Due to a Change in Screen Size

The relationships in the structure of the UI are made explicit using RST. These relationships determine whether an element is either more or less important than another. This knowledge will be used if the elements on the screen do not fit. Firstly, however, UI components will be adjusted in order to try to make the elements fit again. If this fails, the UI components can be transformed into smaller UI components. In this research, these two steps are defined by the UI components and do not make use of discourse knowledge. The last step, however, needs the discourse knowledge. The relationships of the elements to each other will be investigated in order to identify whether some of the elements that do not fit are less important than others. As these elements are, in fact, additional information, they will be hidden in order to make space on the UI. The important elements may never be hidden. For example, Fig. 16a represents the UI of one of the use cases. If the elements no longer fit on the screen, the RST in Fig. 16c will be used. As elements 3 and 4 hold the relationship elaboration, it becomes clear why element 4 is additional information. This element will, therefore, be hidden, as shown in Fig. 16b.

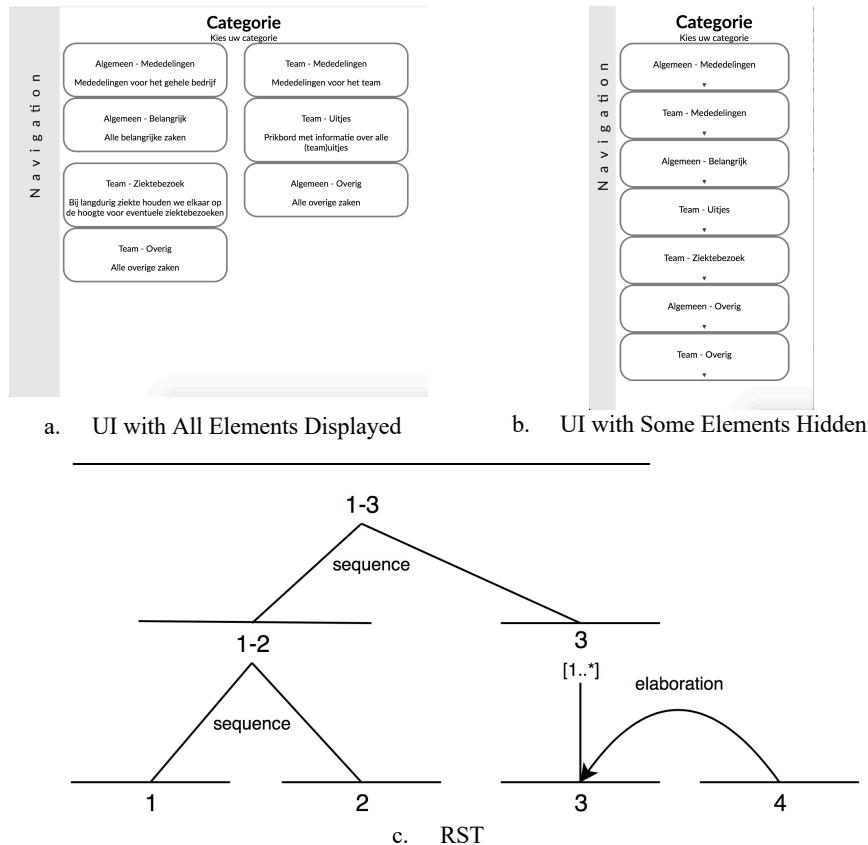


Fig. 16. UIs and RST of a Use Case

This research assumes that all elements should fit on the screen. In reality, however, this may not always be the case. Some elements will only be visible when scrolling. Changing the UI because some elements no longer fit, as discussed in section 7, may not always be necessary.

10 Conclusion & Future Work

In this study we investigated the role of discourse knowledge for generating flexible UIs for application functionalities. The layout of UIs should be variable due to a change in screen size or in user requirements. A framework for generating UIs was set-up to include both domain and discourse knowledge as well as layout information. This framework was applied to three use cases in order to investigate how discourse knowledge is used.

Our research has demonstrated that the presentation structure of the UI can be adjusted using discourse knowledge. The presentation structure represents how the UI elements are related to each other. The layout of each UI element is specified using UI components. For each UI element, multiple possible UI components can be specified. This results in the generation of multiple useful layouts. If a UI component does not fit the screen due to the screen size, either the size of the UI component will be reduced or a custom UI component will be used. If this does not help, then the discourse knowledge is used to identify which UI components can be hidden. In conclusion, discourse knowledge can be used to adjust the presentation structure, to allow multiple layouts of a single functionality and to identify the UI components that can be hidden.

A point for discussion to our approach are the use cases. The use cases do not require a lot of interaction with the user. Additionally, they represent simple functionalities, as the user can understand and perform the functionality easily. Another point for discussion to our approach is the process of identifying the UI elements of the functionality. A domain model is used to identify the UI elements, however, some UI elements do not use any of the information represented in this domain model, such as the title and explanation of the functionality. Therefore, further research can investigate how domain models can contain all information needed to identify all UI elements.

This study provides meaningful knowledge about the role of discourse knowledge for generating flexible UIs. This knowledge can be used for future work. Here, it is important to take into consideration the limitations of this study. More research is necessary to more and other use cases to generalise the findings. Additionally, research of the identification of elements of the UI is needed in order to improve the process.

11 References

- André, E., & Rist, T. (1992). The design of illustrated documents as a planning task. In *Intelligent multimedia interfaces*, ed. M. Maybury, 94-116. New York: AAAI Press.
- Antonio (2015). Free Avatar Pack [Blog post]. Licensed under *Place it Blog*. Retrieved from: <https://blog.placeit.net/free-avatar-pack/> [Accessed 15 May 2018].
- Azar, M. (1999). Argumentative text as rhetorical structure: An application of rhetorical structure theory. *Argumentation*, 13(1), 97-114.
- Bateman, J., Kleinz, J., Kamps, T., & Reichenberger, K. (2001). Towards constructive text, diagram, and layout generation for information presentation. *Computational Linguistics*, 27(3), 409-449.
- Blok, M. C., & Cybulski, J. L. (1998, December). Reusing UML specifications in a constrained application domain. In *Software Engineering Conference, 1998. Proceedings*. (pp. 196-202). Asia Pacific: IEEE.
- Calvary, G., & Coutaz, J. (2014). Introduction to model-based user interfaces. *Group Note*, 7, W3C.
- Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. *Interacting with computers*, 15(3), 289-308.
- Carlson, L., Marcu, D., & Okurowski, M. E. (2003). Building a discourse-tagged corpus in the framework of rhetorical structure theory. In *Current and new directions in discourse and dialogue* (pp. 85-112). Springer, Dordrecht.
- Cranefield, S. (2001). UML and the Semantic Web. *Proceedings of the International Semantic Web Working Symposium*.
- Fakhroutdinov, K. (n.d.). Class. Retrieved October 10, 2018, from <https://www.uml-diagrams.org/class.html>
- Felfernig, A., Friedrich, G. E., & Jannach, D. (2000). UML as domain specific language for the construction of knowledge-based configuration systems. *International Journal of Software Engineering and Knowledge Engineering*, 10(04), 449-469.
- Ferré, X., & Vegas, S. (1999, June). An evaluation of domain analysis methods. In *Proceedings 4th CAiSE Workshop on Exploring Modelling Methods for Systems Analysis and Design*.
- Galitz, W. O. (2007). *The essential guide to user interface design: an introduction to GUI design principles and techniques*. John Wiley & Sons.

- Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubézy, M., Eriksson, H., Noy, N. F., & Tu, S. W. (2003). The evolution of Protégé: an environment for knowledge-based systems development. *International Journal of Human-computer studies*, 58(1), 89-123.
- Geurts, J., Bocconi, S., Van Ossenbruggen, J., & Hardman, L. (2003, October). Towards ontology-driven discourse: From semantic graphs to multimedia presentations. In *International Semantic Web Conference* (pp. 597-612). Springer, Berlin, Heidelberg.
- Höök, K. (2000). Steps to take before intelligent user interfaces become real. *Interacting with computers*, 12(4), 409-426.
- Laubheimer, P. (2016, March 27). Front-End Style-Guides: Definition, Requirements, Component Checklist. Retrieved from <https://www.nngroup.com/articles/front-end-style-guides/>
- Macik, M., Cerny, T., & Slavik, P. (2014). Context-sensitive, cross-platform user interface generation. *Journal on Multimodal User Interfaces*, 8(2), 217-229.
- Maguire, M. C. (1999). A review of user-interface design guidelines for public information kiosk systems. *International Journal of Human-Computer Studies*, 50(3), 263-286.
- Mann, W. C., & Thompson, S. A. (1987). Rhetorical structure theory: Description and construction of text structures. In *Natural language generation* (pp. 85-95). Springer, Dordrecht.
- Modelio (Version 3.7.1) [Computer software] (2018). Modeliosoft. Retrieved from <https://www.modelio.org>
- Morisio, M., Travassos, G. H., & Stark, M. E. (2000, September). Extending UML to support domain analysis. In *Automated Software Engineering, 2000. Proceedings ASE 2000. The Fifteenth IEEE International Conference on* (pp. 321-324). IEEE.
- Mulder, C. (2016). Reducing implementation time for the GUI design-to-code process using DSL.
- Scardamalia, M., & Paris, P. (1985). The function of explicit discourse knowledge in the development of text representations and composing strategies. *Cognition and instruction*, 2(1), 1-39.
- Sobernig, S., Hoisl, B., & Strembeck, M. (2016). Extracting reusable design decisions for UML-based domain-specific languages: A multi-method study. *Journal of Systems and Software*, 113, 140-172.
- Störrle, H. (2013). Towards clone detection in UML domain models. *Software & Systems Modeling*, 12(2), 307-329.

- Taboada, M., & Mann, W. C. (2006). Rhetorical structure theory: Looking back and moving ahead. *Discourse studies*, 8(3), 423-459.
- Thompson, S. A., & Mann, W. C. (1987). Rhetorical structure theory. *IPRA Papers in Pragmatics*, 1(1), 79-105.
- Vairamuthu, S., & Anoucia, S. M. (2018). Design of near optimal user interface with minimal UI elements using evidence based recommendations and multi criteria decision making: TOPSIS method. *International Journal of Humanitarian Technology*, 1(1), 40-65.
- Visser, M. (2011). User requirements and visualizations for a multi-facet search task in the domain of travel (Doctoral dissertation, Information Access).
- Van Ossenbruggen, J., Geurts, J., Cornelissen, F., Hardman, L., & Rutledge, L. (2001, April). Towards second and third generation web-based multimedia. In *Proceedings of the 10th international conference on World Wide Web* (pp. 479-488). ACM.
- Zeldes, A. (2016). RstWeb - A Browser-based Annotation Interface for Rhetorical Structure Theory and Discourse Relations. In *Proceedings of NAACL-HLT 2016 System Demonstrations*. San Diego, CA, 1-5.

12 Appendix

Appendix 1: Functional requirements (section 5)

Use case 1

Teambezetting Overzicht

Voor elke periode in het jaar moet voor een team worden aangegeven hoeveel werkuren een team nodig denkt te hebben om de werkzaamheden die gedaan moeten worden te voltooien. Dit moet worden weergegeven als de minimale bezetting van het team. Om rekening te houden met eventuele uitval door bijvoorbeeld ziekte moet er ook een buffer worden opgesteld om de uitval op te vangen.

Om de minimale bezetting + buffer te halen, moet duidelijk zijn welke werknemers hoeveel uur werken in de periode. In het overzicht is daarom de profielfoto van de medewerkers, voor- en achternaam en het aantal uren te zien. Het totaal aantal uren van alle werknemers in het bezettingoverzicht staat dus gelijk aan de minimale bezetting + buffer. Om een overzicht te krijgen over wat de medewerkers doen in het team wordt ook de functie en de rol van de medewerker in het team weergegeven. Dit is echter wel extra informatie.

Use case 2

Invoeren Verlof

De CAO bepaalt op hoeveel vakantiedagen of vakantieuren een medewerker recht heeft. Het systeem kan beide eenheden (dagen of uren) verwerken. De vakantiedagen zijn voor elke medewerker raadpleegbaar via de verlofapp. Naast de toegekende dagen heeft de medewerker ook inzicht in de nog beschikbare verlofdagen en het aantal verbruikte verlofdagen. De medewerker kan vakantie aanvragen via deze verlof app. Aangezien verlof niet alleen over vakantieaanvragen gaat, moet de medewerker eerst het juiste verloftype kiezen, namelijk vakantie. Vervolgens kan de medewerker de gewenste verlofperiode aanvragen via een kalender-inputveld. Het systeem berekent of het aantal aangevraagde verlofdagen past in het aantal nog beschikbare verlofdagen (1). Daarnaast checkt het systeem of de aanvraag past in de teamplanning (2).

1. Check beschikbaarheid verlofdagen: de volgende uitkomsten zijn mogelijk:

- het aantal aangevraagde verlofdagen is kleiner of gelijk aan het aantal beschikbare verlofdagen. Het systeem geeft automatisch akkoord, mits de teamplanning dat toestaat, zie hiervoor (2);
- het aantal aangevraagde verlofdagen is groter dan het aantal beschikbare verlofdagen; Het systeem geeft geen akkoord voor de aanvraag.

2. Check beschikbaarheid teamplanning: de volgende uitkomsten zijn mogelijk:

- het verlof bevindt zich geheel in een ‘groene’ periode => het systeem geeft akkoord^[SEP]
- het verlof bevindt zich geheel of gedeeltelijk in een ‘amber’ periode, maar niet in een ‘rode’ periode => actie van het team nodig.

- het verlof bevindt zich geheel of gedeeltelijk in een ‘rode’ periode => het systeem wijst de aanvraag af. Voor meer details over rode, amber en groene periodes: zie teamplanning

Het systeem koppelt het resultaat aan de medewerker terug.

Use case 3

Categorie kiezen van prikbord

De werkgever, maar ook de medewerkers kunnen posts aanmaken. Een post bevat “vrije tekst” en wordt op het prikbord geplaatst. De post blijft daar standaard 2 weken hangen en wordt daarna weer automatisch verwijderd.

De posts worden onderverdeeld in categorieën. Indien passend op het scherm staat bij elke categorie een korte samenvatting. De categorieën kunnen alleen voor het team zijn, maar ook voor het hele bedrijf. Dit staat daarom ook aangegeven bij de titel.

Appendix 2: Classes, attributes and relationships for functionalities (section 5).
Use case 1

TEAMBEZETTING BEKIJKEN	
Class	Attribuut
Team	naam
Teambezetting	minimaleBezetting [label;value;units], buffer [label;value;units], datumBegin, datumEinde
Medewerker	voornaam, achternaam, foto
Functie	naam
Rol	naam
Bezetting Accordatie	
Poortwachter	
Verlof Management	
Responsetijd Management	
HR Chat Management	
Relaties	
[Medewerker - Functie 0..1]	
[Medewerker - Rol 0..*]	
[Rollen - Bezetting accordatie 0..1]	
[Rollen - Poortwachter 0..1]	
[Rollen - Verlof Management 0..1]	
[Rollen - Responsetijd Management 0..1]	
[Rollen - HR Chat Management 0..1]	
[Team - Medewerker 1...40]	
[Team - Teambezetting 1]	

Use case 2:

VAKANTIEDAGEN OPNEMEN* **	
Class	Attribuut
Medewerker	voornaam, achternaam
Verlofdagen	totaalAantal, beschikbaarAantal
Vakantie Verlof	datumBegin, datumEinde

Relaties
[Medewerker - Absentiedagen 1]
[Absentiedagen - Verlofdagen 1]
[Verlofdagen - Vakantie verlof 1]

* "Opgenomen uren" = Verlofdagen.totaalAantal - Verlofdagen.beschikbaarAantal

** The system/back-end depends if the application is directly accepted. This information is therefore not included in the domein model

Use case 3

CATEGORIE KIEZEN	
Class	Attribuut
Categorie	naam, samenvatting, toegankelijkheid

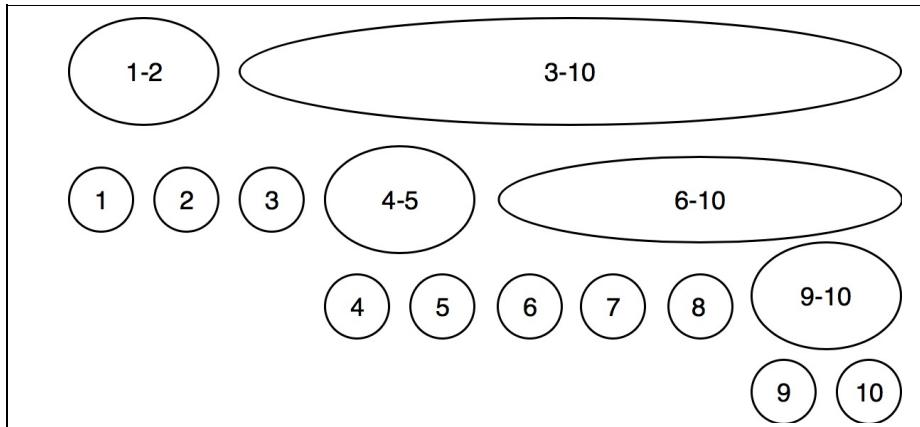
Appendix 3: Adjustments of the domain model (section 5)

Before	After
<p>The diagram shows the initial domain model structure. It includes classes like Prive Verlof, Opleidingsdagen, Verzuimdagen, Langdurig Verlof, Vakante Verlof, Absentiedagen, Zakenreis, and Ledenverlof, connected by various associations with multiplicity 0..1 or 1..1.</p>	<p>The diagram shows the adjusted domain model structure. The 'Absentiedagen' class has been expanded into 'Opleidingsdagen', 'Verzuimdagen', 'Langdurig Verlof', 'Vakante Verlof', and 'Verlofdagen'. Associations are updated to reflect these changes, such as 'Prive Verlof' now connecting to 'Verlofdagen'.</p>
<p>Alle soorten verlof werden onder de class "absentiedagen" gezet. Echter, er zit verschil tussen verlof-, opleidings- en verzuimdagen. Daarom is besloten om deze opsplitsing te maken. In de oude situatie werd onder de class "verlofdagen" alleen vakantiedagen gezien, terwijl dit ook langdurig, prive, bijzonder en kind verlof bevat.</p>	<p>The diagram shows the adjustment of Teamplanning. The original 'Teamplanning' class is split into 'Teambezetting' and 'Teamplanning'. 'Teambezetting' contains attributes for date range and buffer. 'Teamplanning' contains attributes for period and start/end dates. A relationship 'geb' connects them with multiplicity 0..1 on both sides.</p>

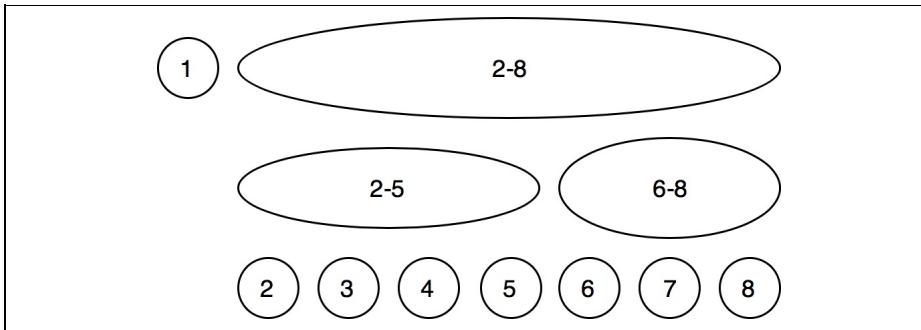
<p>Rol</p> <p>+ naam : string + bevoegdheid ...</p>	<pre> classDiagram class Rol { +naam : string +bevoegdheid ... } class Bezettings_Accorde class Poortwachter class Verlof_Management class Responsetijd_Management class HR_Chat_Management Rol "1..>" "*" Bezettings_Accorde Rol "1..>" "*" Poortwachter Rol "1..>" "*" Verlof_Management Rol "1..>" "*" Responsetijd_Management Rol "1..>" "*" HR_Chat_Management </pre>
<p>De class “Rol” kan bestaan uit 5 andere classes. Daarom is besloten om deze ook toe te voegen in het model met een composite relatie.</p>	
<p>Prikbord</p> <p>+ Post : string</p>	<pre> classDiagram class Posts { +titel : string +content : string } class Categorie { +naam : string +samenvatting : string +toegankelijkheid : ... } Posts "1" --> "1..*" Categorie : + hoort bij </pre>
<p>In eerste instantie was een class aangemaakt voor de functionaliteit “Prikbord”. Echter, deze class is te generiek, aangezien niet duidelijk is wat voor elementen post heeft. Hierdoor is de class aangepast naar “Posts” met daarbij attributen titel en content. Hierbij komt ook de class “Categorie” die nog niet opgenomen was.</p>	
<p>Externe Partij</p> <p>+ naam : string + plaats : string + straatnummer : string + postcode : string + straat : string</p>	<pre> classDiagram class Externe_Partij { +naam : string +plaats : string +straatnummer : string +postcode : string +straat : string } class Leverancier class Overig class Klant { +aantalUur : int ... } Externe_Partij "0..1" --> "0..1" Leverancier Externe_Partij "0..1" --> "0..1" Overig Externe_Partij "0..1" --> "1" Klant : + werk </pre>
<p>De class “Externe partij” kan bestaan uit 3 soorten partijen. Daarom zijn de klassen “Leverancier”, “Overig” en “Klant” toegevoegd. Hierbij is voor “Klant” de attribuut aantalUur toegevoegd, om vast te stellen hoelang er voor een klant gewerkt moet worden.</p>	
	<pre> classDiagram class Opleiding { +naam : string +niveau : string } </pre>
<p>De opleidingen die medewerkers gevolgd hebben moeten ook opgeslagen worden. Daarom is de class “Opleiding” toegevoegd.</p>	

Appendix 4: Elements and structure of functionalities (section 6).

TEAMBEZETTING BEKIJKEN	
Elements	
Structure	
1	Teambezetting
2	{Team.naam}
3	{Teambezetting.datumBegin} - {Teambezetting.datumEinde}
4	Minimale Bezetting: {Teambezetting.minimaleBezetting} uur
5	Buffer: {Teambezetting.buffer} uur
6	{Medewerker.foto}
7	{Medewerker.naam}
8	{Medewerker.aantalContacturen} uur
9	{Functie.niveau} {Functie.naam}
10	{Rol.naam}
1 - 2	Element 1 is de titel van de UI en element 2 de ondertitel. Deze twee elementen kunnen daarom gegroepeerd worden om de (onder)titels te scheiden van de content.
3 - 10	Alle elementen buiten de (onder)titels om zijn de daadwerkelijke elementen van de functionaliteit.
4 - 5	Elementen 4 en 5 geven beide informatie over de minimale bezetting en de buffer waar aan moet worden voldaan. Deze requirements worden daarom gegroepeerd.
6 - 10	Alle informatie over een medewerker wordt weergegeven in de elementen 6 tot en met 10.
9 - 10	Elementen 9 en 10 bevatten beide extra informatie over een medewerker.



INVOEREN VAKANTIEDAGEN	
Elements	
1	Vakantie Langdurig Privé Bijzonder Kinder
2	{Team.naam}
3	Dit jaar heb je recht op calc({Verlofdagen.totaalAantal}/8) vakantiedagen/ {Verlofdagen.totaalAantal} vakantieuren
4	Tot nu toe heb je calc(calc({Verlofdagen.totaalAantal} - {Verlofdagen.beschikbaarAantal})/8) vakantiedagen/ calc({Verlofdagen.totaalAantal} - {Verlofdagen.beschikbaarAantal}) vakantieuren opgenomen.
5	Dit betekent dat er nog calc({Verlofdagen.beschikbaarAantal}/8) vakantiedagen/ {Verlofdagen.beschikbaarAantal} vakantieuren over zijn.
6	input({Vakantieverlof.datumBegin}) - input({Vakantieverlof.datumEinde})
7	Aanvragen
8	In deze periode wordt je aanvraag direct {systemDependApplication}*
Structure	
2 - 8	All elements except the title are the actual elements needed to perform the functionality.
2 - 5	The elements containing information about the holiday of the employee, including the salutation
6 - 8	The elements used for the employee's application



* The system/back-end depends if the application is directly accepted or not

Note that number inside calc(...) will be calculated and the user needs to interact with the elements inside input(...).

CATEGORIE KIEZEN	
Elements	
1	Prikbord
2	Kies uw categorie
3	{Categorie.toegankelijkheid} - {Categorie.naam}
4	{Categorie.samenvatting}
Structure	
1-2	Element 1 is the title and element 2 the subtitle. These two elements can therefore be grouped together to distinguish the (sub)titles from the content.
3 - 4	All elements except the (sub)titles are the actual elements needed to perform the functionality.

The diagram illustrates a hierarchical structure. At the top, there are two large ovals side-by-side, labeled '1-2' and '3-4'. Below them, there is a horizontal row of four small circles, each labeled with a number from 1 to 4.

Appendix 5: Relationships of the elements of functionalities (section 6)

TEAMBEZETTING BEKIJKEN			
Multinuclear Relaties			
Elementen	Relatie	Uitleg	
1-2, 3-10	Sequence	De titel en ondertitel moeten altijd eerder op het scherm te zien zijn dan de content.	
1, 2	Sequence	De titel moet eerder komen dan de ondertitel.	
3, 4-5, 6-10	Sequence	Welke informatie getoond moet worden is afhankelijk van de periode. Daarom wordt deze informatie als eerst weergegeven. Hierna komt de informatie over het aantal uur dat de teambezetting moet halen om te voldoen aan de eisen. De daadwerkelijke invulling om aan deze eisen te voldoen komt hierna.	
4, 5	Joint	De minimale bezetting en de buffer hebben geen volgorde ten opzichte van elkaar. Het maakt daarom niet uit welke informatie voor de andere staat.	
6, 7, 8	Sequence	Voor het weergeven van de informatie van werknemers is gekozen.	
9, 10	Sequence	Gekozen is om eerst de functie te laten zien voordat de rol wordt weergegeven.	
Presentational Relaties			
Nucleus	Satellite	Relatie	Uitleg
6-8	9-10	Elaboration	Zoals vermeld in de functionele requirements is de functie en rol van de medewerker extra informatie.

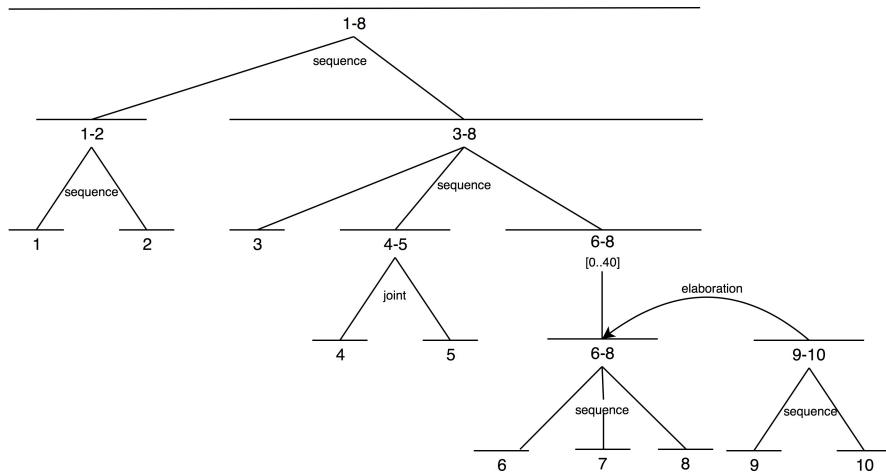
VAKANTIEDAGEN OPNEMEN		
Multinuclear Relaties		
Elementen	Relatie	Uitleg

2-5, 6-7	Sequence	Om vakantie op te nemen moet een werknemer wel vakantiedagen beschikbaar hebben. Deze informatie wordt daarom eerder weergegeven ten opzichte van de mogelijkheid om de aanvraag uit te voeren.	
2, 3, 4, 5	Sequence	De aanhef, informatie over het totaal aantal, opgenomen aantal en beschikbaar aantal vakantiedagen wordt in logische volgorde weergegeven.	
6, 7	Sequence	De invoervelden moeten eerst ingevuld worden voordat een medewerker de aanvraag kan versturen. Daarom worden deze velden eerder weergegeven dan de knop om de aanvraag te voltooien.	
Presentational Relaties			
Nucleus	Satellite	Relatie	Uitleg
1	2-7	Volitional Result	Een medewerker moet kiezen wat voor soort absentie opgenomen moet worden. Voor deze functionaliteit zal dit Vakantie zijn. Deze bewuste keuze zal resulteren tot het weergeven van de elementen om daadwerkelijk een vakantie aan te vragen.
6	8	Volitional Result	Het invullen van de input velden resulteert in feedback over de acceptatie van de aanvraag.

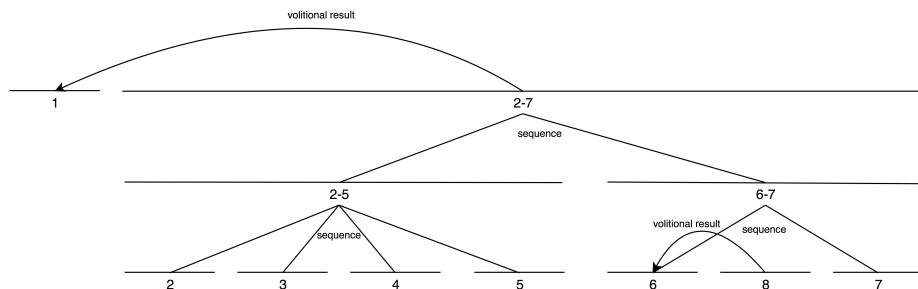
CATEGORIE KIEZEN			
Multinuclear Relaties			
Elementen	Relatie	Uitleg	
1-2, 3	Sequence	De titel en ondertitel moeten altijd eerder op het scherm te zien zijn dan de content.	
1, 2	Sequence	De titel moet eerder komen dan de ondertitel.	
Presentational Relaties			
Nucleus	Satellite	Relatie	Uitleg
3	4	Elaboration	De samenvatting over de categorie wordt gezien als extra informatie over de titel van de categorie.

Appendix 6: RST's of functionalities (section 6)

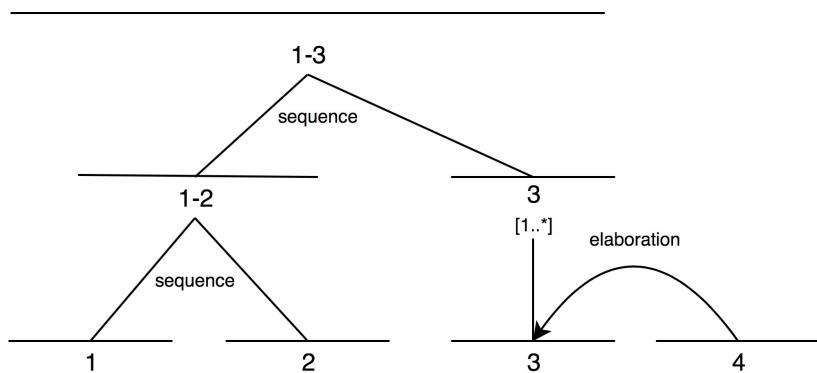
Teambezetting bekijken



Vakantiedagen aanvragen



Categorie kiezen



Appendix 7: XML of RST's (section 6.1)

Teambezetting bekijken

```
<rst>
<header>
  <relations>
    <rel name="elaboration" type="rst"/>
    <rel name="joint" type="multinuc"/>
    <rel name="sequence" type="multinuc"/>
  </relations>
</header>
<body>
  <group id="11"/>
  <group id="12" parent="11" relname="sequence"/>
  <segment id="1" parent="12" relname="sequence">Teambezetting</segment>
  <segment id="2" parent="12">{Team.naam}</segment>
  <group id="13" parent="11" relname="sequence"/>
  <segment id="3" parent="13" relname="sequence">{Teambezetting.datumBegin} - {Teambezetting.datumEinde}</segment>
  <group id="14" parent="13" relname="sequence"/>
  <segment id="4" parent="14" relname="joint">Minimale Bezetting:{Teambezetting.minimaleBezetting} uur</segment>
  <segment id="5" parent="14" relname="joint">Buffer: {Teambezetting.buffer} uur</segment>
  <group id="15" parent="13" relname="sequence" minEntities="1" maxEntities="40" multiplicityId="1" />
  <segment id="6" parent="15" relname="sequence">Medewerker.foto</segment>
  <segment id="7" parent="15" relname="sequence">{Medewerker.naam}</segment>
  <segment id="8" parent="15" relname="sequence">{Medewerker.aantalContacturen} uur</segment>
  <group id="16" parent="13" relname="elaboration"/>
  <segment id="9" parent="16" relname="sequence">{Functie.niveau} {Functie.naam}</segment>
  <segment id="10" parent="16" relname="sequence">{Rol.naam}</segment>
</body>
</rst>
```

Vakantiedagen invoeren

```
<rst>
<header>
  <relations>
    <rel name="elaboration" type="rst"/>
    <rel name="joint" type="multinuc"/>
    <rel name="sequence" type="multinuc"/>
    <rel name="result" type="rst"/>
  </relations>
</header>
<body>
  <segment id="1" >Vakantie Langdurig Privé Bijzonder Kinder</segment>
  <group id="9" type="span"/>
  <group id="10" type="multinuc" parent="9" relname="sequence"/>
  <segment id="2" parent="10" relname="sequence">Hoi {Medewerker.voornaam} {Medewerker.achternaam}</segment>
  <segment id="3" parent="10" relname="sequence">Dit jaar heb je recht op calc({Verlofdagen.totaalAantal}/8) vakantiedagen/ {Verlofdagen.totaalAantal} vakantieuren</segment>
</body>
</rst>
```

```

<segment id="4" parent="10" relname="sequence">Tot nu toe heb je calc(calc({Verlofdagen.totaalAantal} - {Verlofdagen.beschikbaarAantal})/8) vakantiedagen/ calc({Verlofdagen.totaalAantal} - {Verlofdagen.beschikbaarAantal}) vakantieuren opgenomen.</segment>
<segment id="5" parent="10" relname="sequence">Dit betekent dat er nog calc({Verlofdagen.beschikbaarAantal}/8) vakantiedagen/ {Verlofdagen.beschikbaarAantal} vakantieuren over zijn.</segment>
<group id="11" type="multinuc" parent="9" relname="sequence"/>
<segment id="6" parent="11" relname="sequence">input(Vakantie.Verlof.datumBegin) - input(Vakantie.Verlof.datumEinde)</segment>
<segment id="7" parent="6" relname="result">In deze periode wordt je aanvraag direct geaccepteerd!</segment>
<segment id="8" parent="11" relname="sequence">Aanvragen</segment>
</body>
</rst>
```

Categorie kiezen

```

<rst>
<header>
<relations>
<rel name="elaboration" type="rst"
<rel name="sequence" type="multinuc"/>
</relations>
</header>
<body>
<segment id="1" >Prikbord</segment>
<segment id="2" >Kies uw categorie</segment>
<group id="3" parent="6" type="span" minEntities="1" maxEntities="40" multiplicityId="1" />
<segment id="4" parent="3" relname="sequence">Categorie.naam</segment>
<segment id="5" parent="4" relname="elaboration">(Categorie.samenvatting)</segment>
<group id="6"/>
</body>
</rst>
```

Appendix 8: Style guide of the use cases (section 7)

Style Guide

UI component templates

Atomic UI Components

UI Component: **Title**

Variable	Font-size	
Variable	Min. font-size	
Variable	Content	
	HTML	<h1>Content</h1>

UI Component: **Text**

Variable	Font-size	
Variable	Min. font-size	
Variable	Content	
	HTML	<p>Content</p>

UI Component: **TextLine**

Variable	Font-size	
Variable	Min. font-size	
Variable	Content	
	HTML	Content

UI Component: **LabelValueUnitInline**

Variable	Font-size	
Variable	Min. font-size	
Variable	Content	Label
Constant		:
Variable		Value
Variable		Unit
	HTML	<p>Label : Value Unit</p>

UI Component: LabelValueUnitTwoLines

Variable	Font-size	
Variable	Min. font-size	
Variable	Content	Label
Variable		Value
Variable		Unit
	HTML	<p>Label</br>
		Value Unit</p>

UI Component: LabelTwoValuesTwoUnitsTwoLinesInline

Variable	Font-size	
Variable	Min. font-size	
Variable	Content	Label
Variable		Value1
Variable		Unit1
Variable		Value2
Variable		Unit2
	HTML	Label</br>
		Value1 Unit1/ Value2 Unit2

UI Component: Image

Variable	Width	
Variable	Height	
Variable	Min. ratio	
Variable	Shape	
Variable	Source	
	HTML	

UI Component: Period

Variable	Font-size	
Variable	Min. font-size	
Variable	Content	StartDate
Constant		-
Variable		EndDate
	HTML	<p>StartDate - EndDate</p>

UI Component: PeriodInput

Variable	Font-size	
Variable	Min. font-size	
Variable	Content	Label1
Variable		StartDate
Constant		-
Variable		Label2
Variable		EndDate
HTML		<pre> <table> <tbody> <tr> <th> Label1
 <input type="text" id="from" class="date hasDatepicker" name="from"> </th> <th></th> <th> Label2
 <input type="text" id="to" name="to" class="hasDatepicker"> </th> </tr> </tbody> </table> </pre>

UI Component: Button

Variable	Width	
Variable	Height	
Variable	Text	
Variable	Min. ratio	
Variable	Source	
Variable	Font-size	
	HTML	<button src="Source"/>

UI Component: DropdownMultipleOptions

Variable	Font-size	
Variable	Min. font-size	
Variable	Content	[Tab1,...,Tabn]
	HTML	<pre> <select> <option>Tab1</button> <option>Tab2</button> <option>Tabn</button> </select> </pre>

Composite UI Components

UI Component: Composite

Variable	Order	
Variable	Border	
	HTML	<div>AtomicComponents</div>

UI Component: CompositeMultiplicity

Variable	OrderMultiplicity	
Variable	OrderAtomic	
Variable	BorderMultiplicity	
Variable	BorderAtomic	
	HTML	<div> <div>AtomicComponents1</div> <div>AtomicComponents...</div> </div>

Values

Fontsize

SmallBodyFontSize	12px
MainBodyFontSize	18px
LargeBodyFontSize	22px
MainTitleFontSize	30px
MainSubTitleFontSize	26px

Min_Fontsize

NoScaling	100%
MainScaling	80%

Border...

NoneBorder	None
GreyBorder	solid grey

Shape

Normal	100%
Circle	50%

Order

Vertical	100%
Horizontal	(100 / number of atomic UI components)%

Teambezetting bekijken

Atomic UI Components

UI element 1: Teambezetting

Original

UI component		Title
Font-size		MainTitleFontSize
Min. font-size		NoScaling
Content		Teambezetting

Custom

None

UI element 2: {Team.naam}

Original

UI component		Text
Font-size		MainSubTitleFontSize
Min. font-size		NoScaling
Content		{Team.naam}

Custom

None

UI element 3: {Teambezetting.datumBegin} - {Teambezetting.datumEinde}

Original

UI Component		Period
Font-size		MainBodyFontSize
Min. font-size		MainScaling
Content	Startdate	{Teambezetting.startDatum}
	Enddate	{Teambezetting.eindDatum}

Custom

None

UI element 4: Minimale Bezetting: {Teambezetting.minimaleBezetting} uur
Original

UI component		LabelValueUnitInline
Font-size		LargeBodyFontSize
Min. font-size		MainScaling
Content	Label	Minimale bezetting
	Value	{Teambezetting.minimaleBezetting}
	Unit	uur

Custom

UI component		LabelValueUnitTwoLines
Content	Label	Minimale bezetting
	Value	{Teambezetting.minimaleBezetting}
	Unit	uur

UI element 5: Buffer: {Teambezetting.buffer} uur**Original**

UI component		LabelValueUnitInline
Font-size		LargeBodyFontSize
Min. font-size		MainScaling
Content	Label	Buffer
	Value	{Teambezetting.buffer}
	Unit	uur

Custom

UI component		LabelValueUnitTwoLines
Content	Label	Buffer
	Value	{Teambezetting.buffer}
	Unit	uur

UI element 6: {Medewerker.foto}**Original**

UI component		Image
Source		{Medewerker.foto}
Min. ratio		NoScaling
Shape		Circle

Custom

None

UI element 7: {Medewerker.voorNaam} {Medewerker.achterNaam}**Original**

UI Component		Text
Font-size		SmallBodyFontSize
Min. font-size		NoScaling
Content		{Medewerker.voornaam} {Medewerker.achternaam}

Custom

None

UI element 8: {Medewerker.aantalContacturen} uur**Original**

UI Component		Text
Font-size		SmallBodyFontSize
Min. font-size		NoScaling
Content		{Medewerker.aantalContacturen} uur

Custom

None

UI element 9: {Functie.niveau} {Functie.naam}**Original**

UI Component		Text
Font-size		SmallBodyFontSize
Min. font-size		NoScaling
Content		{Functie.niveau} {Functie.naam}

Custom

None

UI element 10: {Rol.naam}**Original**

UI Component		Text
Font-size		SmallBodyFontSize
Min. font-size		NoScaling
Content		{Rol.naam}

Custom

None

Composite UI Components

UI Components 1-2

UI Component	Composite
Order	Vertical
Border	None

UI Components 3-10

UI Component	Composite
Order	Vertical
Border	GreyBorder

UI Components 4-5

UI Component	Composite
Order	Horizontal
Border	None

UI Components 6-10

UI Component	CompositeMultiplicity
OrderMultiplicity	HorizontalOneRow
OrderAtomic	Vertical
BorderMultiplicity	None
BorderAtomic	GreyBorder

UI Components 9-10

UI Component	Composite
Order	Vertical
Border	None

Invoeren vakantiedagen

Atomic UI Components

UI element 1: Vakantie Langdurig Privé Bijzonder Kinder Original

UI component		TabMultipleOptions
Font-size		BodyFontSize
Min. font-size		70%
Border		NoneBorder
Content	Tab1	Vakantie
	Tab2	Langdurig
	Tab3	Privé
	Tab4	Bijzonder
	Tab5	Kinder

Custom

UI component		TabMultipleOptions
--------------	--	--------------------

UI element 2: Hoi {Medewerker.voornaam} {Medewerker.achter-naam}

Original

UI component		Text
Font-size		MainBodyFontSize
Min. font-size		MainScaling
Content		Hoi {Medewerker.voornaam} {Medewerker.achternaam}

Custom

None

UI element 3: Dit jaar heb je recht op calc({Verlofdagen.totaalAantal}/8) vakantiedagen / {Verlofdagen.totaalAantal} vakantieuren

Original

UI component		Text
Font-size		MainBodyFontSize
Min. font-size		MainScaling
Content		Dit jaar heb je recht op calc({Verlofdagen.totaalAantal}/8) vakantiedagen / {Verlofdagen.totaalAantal} vakantieuren

Custom

UI component		LabelTwoValuesTwoUnitsTwoLinesInline
Content	Label	Totaal aantal vakantiedagen/-uren
	Value1	calc({Verlofdagen.totaalAantal}/8)
	Unit1	dagen
	Value2	{Verlofdagen.totaalAantal}
	Unit2	uren

UI element 4: Tot nu toe heb je calc(calc({Verlofdagen.totaalAantal} - {Verlofdagen.beschikbaarAantal})/8) vakantiedagen/ calc({Verlofdagen.totaalAantal} - {Verlofdagen.beschikbaarAantal}) vakantieuren opgenomen.

Original

UI component		Text
Font-size		MainBodyFontSize
Min. font-size		MainScaling
Content		Tot nu toe heb je calc(calc({Verlofdagen.totaalAantal} - {Verlofdagen.beschikbaarAantal})/8) vakantiedagen/ calc({Verlofdagen.totaalAantal} - {Verlofdagen.beschikbaarAantal}) vakantieuren opgenomen.

Custom

UI component		LabelTwoValuesTwoUnitsTwoLinesInline
Content	Label	Verbruikt aantal vakantiedagen/-uren
	Value1	calc(calc({Verlofdagen.totaalAantal} - {Verlofdagen.beschikbaarAantal})/8)
	Unit1	dagen
	Value2	calc({Verlofdagen.totaalAantal} - {Verlofdagen.beschikbaarAantal})
	Unit2	uren

UI element 5: Dit betekent dat er nog calc({Verlofdagen.beschikbaarAantal}/8) vakantiedagen/ {Verlofdagen.beschikbaarAantal} vakantieuren over zijn.

Original

UI component		Text
Font-size		MainBodyFontSize
Min. font-size		MainScaling
Content		Dit betekent dat er nog calc({Verlofdagen.beschikbaarAantal}/8) vakantiedagen/ {Verlofdagen.beschikbaarAantal} vakantieuren over zijn.

Custom

UI component		labelTwoValuesTwoUnitsTwoLinesInline
Content	Label	Beschikbaar aantal vakantiedagen/-uren
	Value1	calc({Verlofdagen.beschikbaarAantal}/8)
	Unit1	dagen
	Value2	{Verlofdagen.beschikbaarAantal}
	Unit2	uren

UI element 6: input({Vakantieverlof.datumBegin}) - input({Vakantieverlof.datumEinde})

Original

UI component		PeriodInput
Font-size		MainBodyFontSize
Min. font-size		NoScaling
Content	Label1	Startdatum
	Value1	{Vakantie Verlof.datumBegin}
	Label2	Einddatum
	Value2	{Vakantie Verlof.datumEinde}

Custom

None

UI element 7: Aanvragen**Original**

UI component		Button
Image		/
Text		Aanvragen
Font-size		MainBodyFontSize
Min. ratio		NoScaling

Custom

None

UI element 8: In deze periode wordt je aanvraag direct {systemDependApplication}**Original**

UI component	Text
Font-size	MainBodyFontSize
Min. font-size	MainScaling
Content	In deze periode wordt je aanvraag direct {systemDependApplication}* Content

Custom

None

Composite UI Components

UI Components 2-8

UI Component	Composite
Order	Vertical
Border	GreyBorder

UI Components 2-5

UI Component	Composite
Order	Vertical
Border	None

UI Components 6-8

UI Component	Composite
Order	Vertical
Border	None

Categorie kiezen

Atomic UI Components

UI element 1: Prikbord

Original

UI component		Title
Font-size		MainTitleFontSize
Min. font-size		NoScaling
Content		Prikbord

Custom

None

UI element 2: Kies uw categorie

Original

UI component		Text
Font-size		MainSubTitleFontSize
Min. font-size		NoScaling
Content		Kies uw categorie

Custom

None

UI element 3: {Categorie.toegankelijkheid} - {Categorie.naam}

Original

UI Component		Text
Font-size		MainBodyFontSize
Min. font-size		MainScaling
Content		{Categorie.toegankelijkheid} - {Categorie.naam}

Custom

None

UI element 4: {Categorie.samenvatting}

UI Component		Text
Font-size		SmallBodyFontSize
Min. font-size		MainScaling
Content		{Categorie.samenvatting}

Custom

None

Composite UI Components**UI Components 1-2**

UI Component	Composite
Order	Vertical
Border	None

UI Components 3-4

UI Component	CompositeMultiplicity
OrderMultiplicity	HorizontalAllRows
OrderAtomic	Vertical
BorderMultiplicity	None
BorderAtomic	GreyBorder