A A R H U S  U N I V E R S I T E T


# Master's thesis in Computer Science

## Requirements processing for Web based information systems: guidelines and taxonomy

**Student: Renata Norbutaite**

**Supervisor: Søren Christensen**

**30 April 2009**

# TABLE OF CONTENTS

# ABSTRACT

Every day there are more and more Web based information systems out there. Netcraft claims that the total number of Web sites in general has doubled in the past two years. Applications are getting more complex and even all sorts of businesses are implemented only by means of these systems. One can look at such major examples as an American electronic commerce company Amazon.com which is considered to be the largest online retailer in America, or eBay.com – an online auction and shopping website which gathers today about 233 million members. These systems are as important as those existing in the real world and not less complicated. One of the steps in creating a successful Web information system is using conceptual modelling in a way to fulfil the system's purpose. Building such a system is like building a house. Usually one does not start building a basement without having blue-prints of the whole building. One must know the structure before starting to work. This is not different with Web systems. The thing with conceptual modelling techniques that are applied to Web information systems is that there are so many different solutions for the same concerns. This is basically because there are many diverse approaches towards what a Web information system is. It can be viewed as navigable source pages, or as a complicated data structure, or even as an object-oriented construction. It is not that one approach is wrong or right in its sense; just there are cases when it is hard to know which one fits. Moreover, it is usually unclear which particular approach to use or how to use it. A seemingly undemanding part of modelling as navigation can be expressed as a simple UML diagram using stereotypes, or it can be expressed by a more targeted methodology that OO-H propose. And this is not all that could be done with the aspect of navigation. Nevertheless, there is not one path to follow in order to model a Web information system. There are many of them. There are not enough precise instructions, though. Conceptual modelling falls under a category of requirements processing. It is a mean that helps to visualize the whole system by providing models and schemas and in this way allows the stakeholders and system designers to reach a better consensus. Therefore, this thesis presents an overview of the different solutions, discusses strong and weak points, and, moreover, proposes an approach towards relationships among them. The thing is, when it comes to processing requirements of Web based information systems with means of conceptual modelling, it is important for one to see the most relevant options. What is more, it is important to see advantages and disadvantages of different alternatives, and comparison of similar resolutions. This thesis provides with guidelines to choose those techniques that are the most appropriate for certain purposes, at the same time, it provides the awareness of the limitations of these techniques. Along the way of modelling processes, ideas are supported with practical examples. At the end of this thesis taxonomy of all the discussed solutions is presented.

*'It's not enough that we do our best;*
*Sometimes we have to do what's required.'*
*W. Churchill*

The words of W. Churchill are indeed what concerns many system developers, designers, and architects – putting all the best efforts in making the system as good as possible is quite useless if that system does not fulfil its purpose. A thorough taking care of requirements contributes to that purpose significantly. There are many different techniques for requirements gathering and analysis suggested. Some are as primitive as a few interviews with a customer (only on rare cases this is enough); others can be as complicated as building an entire framework of systems architecture including all stakeholders, analyzing in detail all functional and non-functional requirements and, therefore, spending lots of time so the product would satisfy all interested parties as much as possible.

None the less, there does not seem to be a standard way for requirement analysis or processing. And it would be hard to construct one as there are numbers of different kinds of systems and different developers may adopt different techniques for the same issues. In general, on the one hand, this work of software development is considered to be important but usually not as important as implementation itself. This serves more to a system-centric approach. On the other hand, the need to actually please the user is highlighted these days more than ever. There are many similar products, so it is reasonable to think that the user would choose or prefer the one which satisfies ones needs the most. User's (or the customer's) needs is what requirements are talking about. So as a user-centric approach is what we must have in mind when creating a system, some more structured regulations for requirements would benefit for that matter.

This thesis focuses mainly on Web based information system modelling; and it concentrates on requirements due to this kind of systems. It is important to say that dealing with requirements in general and dealing with requirements for some specific kinds of systems (Web based systems in our case) is not the same thing at all. Let's see what the overall ideas suggest: for example, Ralf R. Young in his *Effective Requirement Practices*[12] gives all the requirements mechanism starting from gathering a responsible team and finishing with suggestions of how to accommodate requirements changes; Karl E. Wiegers in *Software Requirements, Second Edition*[13], alternatively, concentrates more on typical situation analysis, providing examples, and giving sophisticated tips. All these theories are undeniably very relevant, but in specific situations there is more to know. The goal of this thesis is to structure those specific ideas and techniques which mainly are relevant to Web based information systems, and which are a nice complement to overall ideas discussed in these two books.

The overall idea of the thesis is to study papers suggesting similar and different techniques towards requirements processing, to analyze and compare them. The set of methodologies presented is chosen because every of them gives a unique solution towards the problem. And of course, one could find more techniques for this matter out there, but the given ones seem to contribute to the overall thesis goal the most. What is more, a new approach towards collections

of suggestions and relationships among them is presented. So to be clear, this thesis does not propose any new techniques as such but gives a new insight into already existing ones.

## ACKNOWLEDGEMENTS

I would like to thank all who have contributed to achieving the final goal – writing this master's thesis. First of all, I thank my supervisor Søren Christensen for the wonderful supervising and priceless advises while writing this thesis. I thank Olivier Danvy for great series of lectures concerning preparation for master's thesis writing. I also want to thank everyone who helped to find relevant books, papers, and researches concerning the topic of my thesis. Finally, I want to thank all the system administrators for doing their job very well and letting use internet and local area network systems without interruptions.

# 1. TYPES OF REQUIREMENTS

Before going into the essence of the thesis, it makes sense to talk a little about requirements in general. On the whole, the 'requirements' concept covers a big spectrum of different things. This part of the thesis will clarify ideas behind and explain where the focus will be settled.

As it is given in I. Sommerville *CS2 Software Engineering note 2*[20]:

'Requirements are descriptions of the services that a software system must provide and the constraints under which it must operate'.

What is more, this thesis summarizes types of requirements that one might consider. From one point of view, there can be *user requirements*, *system* requirements, and *software specifications*. From the other point of view, there are *functional*, *non-functional*, and *domain* requirements. The table 1.1 below gives definitions of these concepts as summed-up in [20].

**Table 1.1.** ***Types of requirements***

| Stages of evolution | | Foundation | |
|---|---|---|---|
| **User requirements** | Statements in natural language plus diagrams of the services that the system provides and its operational constraints. | **Functional requirements** | Statements of services that the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. |
| **System requirements** | A structured document setting out a detailed description for the system services. | **Non-functional requirements** | Constraints on the services or functions offered by the system such as timing constraints on the development process, standards, etc. |
| **Software specifications** | A detailed software description which can serve as a basis for a design implementation. | **Domain requirements** | Requirements that come from the application domain of the system that reflect the characteristics of that domain. |

As it can be seen from the table, there are some evolutionary steps that requirements take during the process of software creation. Everything starts by talking with users and all the relevant stakeholders in order to figure out what exactly they want the system to do. These user requirements are processed and become system requirements which are later transformed into software specifications. With regards to this practice, the thesis focuses on user requirements processing into system requirements. And in our case, it is user requirements processing into Web based information systems requirements.

With respect to the different aspects of the system, requirements are functional if they indicate what kinds of functions the system provides; or non-functional requirements if they talk about constraints on the system's functionality. The former one is basically what this thesis is all about, although, there are some thoughts with regards to the latter type of requirements as well. These two categories will be discussed a bit more extensively in this part. Domain requirements are somewhat discussed in this thesis as well. Generally, it is Web based systems we are talking

about, therefore, aspects as business modelling requirements, or Web service modelling requirements necessitate for a separate discussion.

In general, when talking about requirements [12] suggests some ideas to help recognizing if it is a good requirement we are talking about. So a requirement has to be necessary, verifiable, attainable, unambiguous, complete, consistent, traceable, allocated, concise, implementation free, standard constructs, and a unique identifier. This given set does not suggest anything concrete but it is beneficial to have these epithets in mind.

## 1.1. FUNCTIONAL REQUIREMENTS

As it is nicely expressed in R. Malan and D. Bredemeyer *Functional Requirements and Use Case*[21]:

'Functional requirements capture the intended behaviour of the system. This behaviour may be expressed as services, tasks or functions the system is required to perform. '.

There are no common properties that would summarize what to expect from functional requirements. For different systems they can be totally different but there are some ideas to suggest if we talk about Web based information systems. First of all, what kind of information is processed in the system, and second of all, the way in which that information can be accessed and handled (referred to navigation). Other properties depend on a particular system.

When it comes to start working with requirements, the usual tactics is to gather statements or sequences of statements from stakeholders. The latter is referred to be so called *scenarios* – sequences of steps one has to perform in order to achieve some functional goal. To make things clear, this is always the first step to do – it is so called requirements gathering. But having statements and scenarios on hand is hardly enough foundation for designing a good system. They have to be processed to get a form of system requirements, as given in [20]. Or in other words, it is important to figure out the *real* user requirements, as [12] indicates. It seems that many papers acknowledge use case modelling technique (taken form UML found in G. Booch, J. Rumbaugh, and I. Jacobson *Unified Modeling Language User Guide, The (2nd Edition)*[16]) for expressing functional requirements in general. In reality, it is a little part of everything that can be done. And this thesis goes beyond this technique. The requirements processing is a very important part and has to be done very thoroughly. We will give a whole separate part of this thesis in order to explain the 'why'.

Anyway, first things first. Narendra, N. C. and Orriens, B. in *Modelling web service composition and execution via a requirements-driven approach*[18] suggests some ideas that we can adopt here while talking about statements and scenarios of functional requirements. Although [18] primarily aims the method for domain requirements, Web service modelling if to be specific, the idea helps to write better requirements for Web based systems in general. So, when giving statements, one has to consider questions as *what*, *who*, *when*, *where*, and *how*, as [18] suggests. Ideas are summarized in the table 1.2.

**Table 1.2.** ***Questions for requirements***

| Question | Indication |
|---|---|
| **what** | A description of some certain functional requirement. |
| **who** | The parties that participate in performing a particular function. |
| **when** | Conditions under which a functional requirement has to be met. |
| **where** | Locational information associated with a requirement. |
| **how** | The way to fulfil a requirement (one might think of a detailed scenario). |
| **why** | A rationale for a requirement to exist at all. |

The table sums-up all the relevant information that can be given with respect to a functional requirement. Usually, stakeholders are not capable to provide such extensive information and that is not always necessary. Nevertheless, this is a thorough starting point. And this thesis, while analyzing different approaches for requirements processing, will always assume that there is at least a subset of this information about a functional requirement. So the big question is figuring out how to handle the information we have in the best possible way.

## 1.2. EXTRA-FUNCTIONAL REQUIREMENTS

Extra-functional requirements (commonly known as non-functional requirements) are sometimes forgotten or taken as not so important as functional requirements. This approach can hurt a project in the same way as not taking into account functional ones. What is more, this kind of requirements is even trickier in a way that they possibly are not explicitly given by stakeholders. This is because one might think that the system has to perform well and be secure even without saying. And, of course, this is a false assumption.

We will borrow the definition of extra-functional requirements from R. Malan and D. Bredemeyer *Defining Non-Functional Requirements or System Qualities*[22]:

'A system has properties that emerge from the combination of its parts. These emergent properties will surely be a matter of accident, if the non-functional requirements, or system qualities, are not specified in advance'.

There is no one correct way to classify all the extra-functional requirements. Different papers give different opinions for this matter and even different names for the same properties. For example, [20] gives a classification according to *product*, *organizational*, and *external* requirements. The table 1.3 below gives the result of this classification.

**Table 1.3.** ***Classification of requirements***

| Product requirements | Organizational requirements | External requirements |
|---|---|---|
| **Portability** | Delivery | Ethical |
| **Reliability** | Implementation | Interoperability |
| **Usability** | Standards | Legislative: Safety<br>Privacy |

| | |
|---|---|
| **Efficiency:** | Space<br>Performance |

If to compare, [22] suggests classification with regards to *run-time*, and *development-time* non-functional properties. The table 1.4 below summarizes this approach.

**Table 1.4.** ***Classification of requirements***

| Run-time | | Development-time | |
|---|---|---|---|
| **Usability:** | Ease-of-use<br>Learnability<br>Memorability<br>Efficiency<br>... | Localizability: | Adaptations due to regional differences |
| **Configurability and supportability** | | Modifiability or extensibility: | Unspecified future functionality |
| **Correctness, reliability, availability** | | Evolvability: | Support new capabilities<br>Exploit new technologies |
| **Quality of service requirements (performance):** | Throughput<br>Response time<br>Transit delay<br>Latency<br>... | Composability: | Compose plug-and-play components |
| **Safety:** | Security<br>Fault tolerance<br>... | Reusability: | Reuse in future systems |
| **Operational scalability:** | Sups port for additional users or sites<br>... | | |

As it can be seen from the tables, there is no one concrete way to define these requirements, or to classify them, and not to mention, there is no standard way to handle them. But there is a bright side here. If these extra-functional requirements are well defined in advance, methods can be applied in order to check if they hold. This is usually done by prototyping a system, so the values of these properties would be checked before overall architecture is build and much money paid.

Again, there are many suggestions of how extra-functional requirements can be expressed. There is a thought in [22] that it is enough just to write descriptions of non-functional requirements together with the use-case diagrams, while modelling functional ones. Now [18] again suggests to, first of all, answer the most relevant questions that can be asked about any requirement: *what*, *who*, *when*, *where*, and *how*. The table 1.5 below summarizes where to draw the attention.

**Table 1.5.** ***Questions for requirements***

| Question | Indication |
|---|---|
| **what** | A description of some certain extra-functional requirement. |

| | |
|---|---|
| **who** | The parties to which that particular requirement is relevant. |
| **when** | Conditions under which an extra-functional requirement has to be met. |
| **where** | Locational information associated with a requirement. |
| **how** | The way to fulfil a requirement (one might think of a detailed scenario). |
| **why** | A rationale for a requirement to exist at all. |

There is one big difference between functional and extra-functional requirements. In the perfect world it is possible to satisfy all the functional requirements but that is no way possible to do with extra-functional ones. For example, a high level of security on the system will certainly hurt the level of performance. Due to this matter, there has to be a way to find a balance in the system in order to figure out those exact extra-functional requirements that would satisfy all the stakeholders. There is so called ATAM method (Architecture Tradeoff Analysis Method) that helps to find a solution, and it is presented in R. Kazman, M. Klein, P. Clements *ATAM: Method for architecture evaluation, Technical Report*[19]. The method includes a number of thoroughly defined steps but what is the most important here to us – the key ideas allowing settling on the right set of extra-functional requirements concerning the overall architecture of the system. We can use a term 'architecture' here as the given definition of these requirements talk about properties that emerge from the combination of system's parts which basically indicate that it is architecture that we are talking about. What is more, this method implements ideas expressed in [12], where we crave for the real system's requirements.

The idea of ATAM is, first of all, to present preliminary desired architecture with solutions suggesting ways of achieving that. Then it is up for an architect to analyze everything and find in the system all the possible *risks*, *sensitivity*, and *tradeoff points*. Risks are points in the system that may be potentially problematic; sensitivity and tradeoff points are key architectural decisions. The former ones are properties affecting one non-functional attribute, whereas, the latter ones affect more. Some negotiations about the tradeoffs necessarily have to be done. Next thing is to gather all scenarios from stakeholders concerning extra-functional requirements, which can be categorised as being *use case*, *growth*, and *exploratory scenarios*. In addition, it was mention before that it is not possible so that all of the scenarios would be equally satisfied. That is why ATAM suggest stakeholders to make scenario ratings in order to clearly figure out priorities hence everyone would know what to expect in the final product. This idea is supported by [12] as well. ATAM technique is not the only one for similar matters but this solution is satisfactory here for us. At this point we were looking for a way to settle with a set of extra-functional requirements and ATAM surely helps to do that.

Now, it can be clearly seen that functional and extra-functional requirements cover different aspects of the system. An extra-functional requirement by no means can be 'just' satisfied. For example, it would be a false to claim that a system is secure. The only thing that can be said is to which level it is secure, and measurements of what exactly security means in a particular system can be provided. Basically, there are no techniques that would allow to model extra-functional requirements. Only a system can be modelled in a way that would reveal some architectural properties. All the measurements are left for system prototypes. Although papers and even own experience suggests that in real life there are still many cases that these measurements are left to be done with the final product; so anything that helps to predict the architectural behaviour in advance is worth doing.

# 2. IDEAS BEHIND REQUIREMENTS

When there is a system to be created, there is a part of a job where one has to deal with customer's (user's) requirements. That is to gather them, to process them and to fulfil them. As it was already mentioned, the common understanding is that dealing with requirements explicitly is not so much important as the rest of the development process. Nevertheless, practice shows that this kind of thinking might cause a lot of trouble with regards to resources, time, and money. The easiest part, of course, is to gather the requirements, but the challenge begins when one has to 'decipher' them in order to understand what stakeholders really need and want the system to do as opposed to what they think and say.

One has to remember, that usually stakeholders for whom the system is being developed are experts in very different areas than system developers and they understand things and the logic of what the system has to do very differently from those who actually will work on it. So, it is up for the development team to discover all the things needed so the system would be created in correct fashion at once.

This thesis concentrates on ideas of requirements processing when it comes to Web information systems. There isn't exact system classification, but if one develops a distributed system, or a reactive one, things that have to be discussed are very different from those that Web information system needs. And as this part of work is undervalued in so many cases, that there is a need to talk more about it.

And in order to give a stronger argument of *why* the requirements processing is important, there are some ideas discussed from [12]. In his book Ralph R. Young talks about all the stages of dealing with requirements and gives many ideas of how to make them as effective as possible, and this thesis is a tiny practical part of what R. Young suggests.

Let's start with a statement from the book that summarizes some of the thoughts above: 'The requirements provided by customers (*stated requirements*) are not the *real requirements*. Additional analysis is required to determine real customer needs and expectations. Then the requirements need to be clarified and restated'. In other words, after the requirements are gathered (stated ones) they have to be processed by system developers and given back to the stakeholders to be verified. It is important that common consensus would be reached between those who create the system and those to whom the system is being created. What is more to say to that: 'A typical project generates issues between the customer and the supplier because of their different perspectives'.

There is one more concept with regards to the requirements. Sometimes the stated requirements are the real requirements. This is what the R. Young in his book has to say: 'Each individual or organization determined to be a customer of the process has expectations of it. Those expectations that are considered to be reasonable by the designers of the process are termed customer valid requirements. These are the requirements of the process that must be met by the successful performance or execution of the process.' So the overall idea is to take stated requirements, process them in order to get the real requirements and finally to make them to be customer valid requirements. The following figure 1.1 illustrates this process.


**Figure 2.1.** ***Process of requirements processing***

The stated requirements mentioned above usually are simple statements or scenarios of what a system has to do and in which way. Of course, some requirements are self understandable and some of them are not so much. And if a designer has a slightest doubt about a requirement, there are some questions, as [12] gives to be asked:

- What exactly do you need?
- Why do you need it?
- How are you going to use it?
- What problems are you having with the existing system? (If there is one, of course)
- What needs to be changed?

Getting the additional information allows a better understanding of how the customer thinks and helps realizing the overall underlying logic of the whole system. This is similar to what we already talked before, specifically, about functional and non-functional requirements where we needed some more extensive information. These ideas were borrowed from [18].

Now there is a bit of statistics that [12] presents. So it would be clearer, the table 2.1 below summarized the main ideas.

**Table 2.1.** ***Statistics***

| Situation | Value |
|---|---|
| The gathered data suggests that in order to achieve the best results, one has to invest in the overall requirements process: | 8% to 14% of total project cost |
| If no formalities are used for the requirements process (as simple as scenarios or use cases), in the testing phase lots of efforts are wasted: | 16% of the test cases are redundant and 10% are irrelevant |
| In the requirements definition stage of product development are the majority of product defects inserted: | More than 80% of all the product defects |
| A 30% change in requirements during the system life cycle will double the cost of the program! | |

What we get from this table is basically that one has to invest a lot in the requirements process; not to avoid formalities so the testing would be efficient; be extremely careful when reaching the consensus with the customer because it is the biggest source of defects that could possibly exist, since else wise the cost grows enormously rapid. With respect to this thesis, it will provide with the necessary set of formalism to be used with requirements in order to make things better.

One more thing to mention with respect to errors (we already know that the majority of them comes with requirements), there is some statistics in table 2.2 of how much it relatively costs to fix an error when it is detected.

**Table 2.2.** ***Error cost***

| Phase in which the error is found | Cost ratio |
|---|---|
| **Requirements** | 1 |
| **Design** | 3-6 |
| **Coding** | 10 |
| **Development testing** | 15-40 |
| **Acceptance testing** | 30-70 |
| **Operation** | 40-1000 |

All these arguments seem to convince that it is important to spend money and time for requirements and to do everything that the requirements would be correct as soon as possible. And this thesis will contribute to that by providing some formalism of how the requirements can be processed in Web based information systems by giving guidelines and taxonomy of the proposed techniques. As it is given in [12]: 'Requirements should be documented graphically and textually and should be made visible to the stakeholders '. There are so many different aspects of the Web based systems, and to summarize requirements only to use case models (suggested as one of the best techniques) would be simply unfair. There is a need to analyze many different and alternative solutions. In the R. Young's book *modelling* activity is suggested as a separate technique for finding the real requirements. And in fact, this thesis is based on this particular technique. But no matter of what is done, it is important to remember that there is always a possibility for requirements to change (that may depend on other things than the customer or the designer).

Nevertheless, this thesis will cover only a tiny part of what the word 'requirements' as a phase has in mind. One who is interested in the whole process should definitely go and read the ideas that R. Young has to give in [12]. In order to share some more insight to this issue, some ideas about alternative requirements techniques, expected changes and the whole process are given in the appendix A.

# 3. FROM REQUIREMENTS GATHERING TO DESIGN APPROACHES: AN OVERVIEW

Whenever there is a need to create a Web based information system, the first thing to do is to gather the requirements and process them so the system would satisfy the customer. The importance of dealing with the requirements in the correct fashion we already discussed in the part 2. As Ceri, S., Fraternali, P., and Matera in *Conceptual Modeling of Data-Intensive Web Applications*[6] suggests, there are 5 main types of Web systems distinguished:

- Commerce sites (business model is to sell some object)
- Content sites (business model is to give information about some object)
- Service sites (business model is to give some object as a service)
- Community sites (business model is to build 'socially shared' objects)
- Context sites (business model is to help locating some objects)

While these types of Web systems are referred to as build upon some business model, it is worth mentioning, that in common practice when talking about Web business one refers mostly to commerce sites. Regardless of different goals, these systems have most things in common. Therefore, when talking about requirements and processing of requirements, there is no need to present different solutions for different types of the systems. In the end, there are some additional words to say separately for commerce and service sites. Other than that, techniques can be applied regardless this difference.

This part of the thesis gives an introduction to the presented ideas, explains the relation between requirements and conceptual modelling, and puts the whole matter into the framing of overall Web system's development process.

Before going into more details, it is worth mentioning that this thesis does not foresee any future types of the Web systems and does not invent any methodologies. The overall idea is to analyse and compare different methods of already existing approaches towards already existing types of Web based information systems.

## 3.1. DESIGN APPROACHES

Web based information systems, as they started to appear, were basically made of a set of Web pages providing some sort of information. Therefore, the main issues to talk about were source pages and navigation among them. As now Web systems developed, there are more aspects to talk about. Web applications are considered somewhat more up to the top of complexity scale in comparison with basic Web information systems. This is due to the idea that operations, as they are understood, in Web applications are more than navigation and information producing. Two approaches in L. Baresi, F. Garzotto and P. Paolini *From Web Sites to Web Applications: New Issues for Conceptual Modeling* [3] are given towards what Web applications exactly mean.

Approach no. 1: “A Web application is regarded as an extension to the notion of a traditional information system, taking into account the need to incorporate navigation and complex information structures.”

Approach no.2: “A Web application is regarded as an extension to a traditional Web site, taking into account the need for incorporating various kinds of application operations.”

As it can be seen, both of these approaches argue that a Web application is an extension to what is considered to be a traditional Web information system. As far as conceptual modelling goes, most approaches were using 2-dimentional Web site model, as it is summed up in [3]. That is basically an aspect of information and an aspect of navigation. Information is essentially an output of the Web system – the content of the Web pages; navigation is, on the other hand, an ordered way of accessing relevant information – paths that a user has to follow. This 2-dimentional design space is illustrated in figure 3.1.

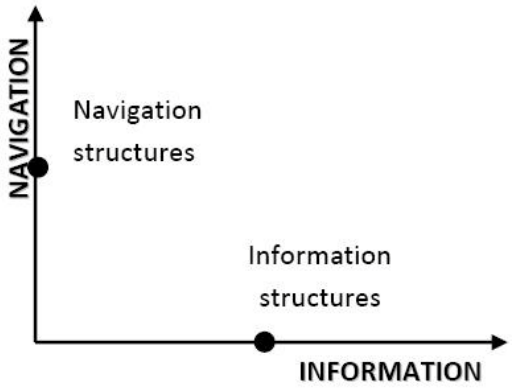



**Figure 3.1.** ***2-dimentional design space***

What is not considered in the information-navigation approach, and what now became relevant when talking about Web applications is an operational aspect. Operations are suggested by [3] as a third dimension, which gives dynamics for both information and navigation. This is illustrated in figure 3.2.

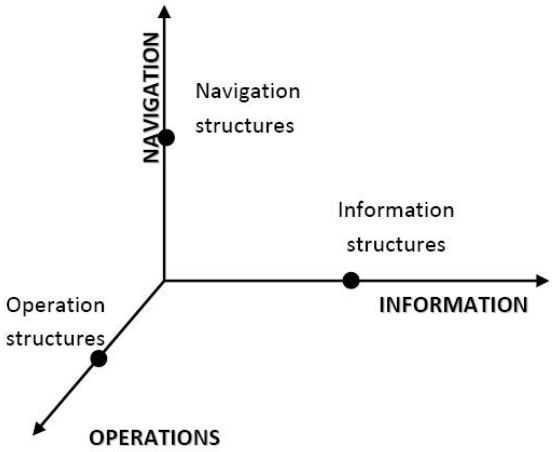



**Figure 3.2.** ***3-dimentional design space***

There are two types of operations commonly distinguished in Web applications: these are *user operations* and *system operations*. User operations are functions that are visible to users,

whereas system operations are triggered by user operations, by administrator operations, or by navigation actions and they are not directly visible by users.

As it is suggested in [3], systems operations are in a lower abstraction level than user operations. Although they are handling semantics of user operations, they are not perceivable by users and therefore might as well be handled in some other phase, and not necessarily during conceptual modelling.

User operations come with two approaches: they can be attached to information 'objects', basically anything that concerns any piece of data in the Web page, or they can be attached to navigational structures – links.

Putting Web systems in a 3-dimentional perspective (information – navigation – operations) gives a clearer picture of the structure and this distinction will come in handy when we move further through different aspects of conceptual modelling.

Now if to look for somewhat different approaches, Gómez, J., Cachero, C., and Pastor, O in *Conceptual Modeling of Device-Independent Web Applications*[4] provides with one. Three basic pieces are considered to be: structure, behaviour, and presentation. Considered method (object-oriented hypermedia) suggest two additional views to the ones captured by traditional modelling: a navigation view and a presentation view which uses elements regarding interface appearance and behaviour in order to model template structures.

One more generalization, as summed up in Schewe, K. and Thalheim, B. *Conceptual modelling of web information systems*[11], is to use a triplet of: content, navigation, and presentation. That would directly lead to database modelling, hypertext structures, and page layout. In the same [11], one more approach is suggested that is oriented towards abstraction layers and design of structure, operations, and interfaces.

As it can be seen, there are quite enough various approaches to choose from. It is not the case that some are better or worse than the others, they are just different. And, as long as all the requirements are satisfied and the customer is happy, there is a difference in which one to use only in perspective of time and resource consumption. The majority of these approaches are analyzed in this thesis and, after discussing them, it will become clearer when to choose what.

In general, this thesis talks about 5 different approaches: information modelling, navigation modelling, operation modelling, behaviour modelling, and presentation modelling. Separately, it also talks a little bit about enterprise modelling, business modelling, transaction modelling, and service modelling. So these techniques wouldn't be so unfamiliar, the following table 3.1. gives some taste of what is all about.

**Table 3.1.** ***Requirements processing techniques***

| Technique | Ideas behind |
|---|---|
| **Information modelling** | It is all about informational content and data structures in the Web site (all the information that appears on the site and the one that does not appear but is contained in the database). |
| **Navigation modelling** | Navigation model at the very minimum shows how to navigate from one site page to the other; navigation can be done within content, data structures; it can be related to operations or transactions. |

| | |
|---|---|
| **Operation modelling** | Under this category falls the idea of expressing how an activity or a set of activities are preceded in the system. It is more like task modelling, and has to be not mistaken with an execution of the service operation. |
| **Behaviour modelling** | The main idea of behaviour modelling is to show what the system reaction to a certain action of a user is. |
| **Presentation modelling** | Presentation model expresses how the information and design details will be placed in the web site; navigation sense can be captured as well. |
| **Enterprise modelling** | This is not exactly a modelling technique, more like some suggestions of how to use modelling techniques in an enterprise case. |
| **Business modelling** | This relates more to commercial systems. Unlike other kinds of systems, commercial ones usually trade actual tangible values. |
| **Transaction modelling** | Transactions are interesting because they absorb a set of operations which has to act as one unit. |
| **Service modelling** | Services in a Web system give an idea that there is part of functionality that is provided by a different supplier. |

Other than these modelling techniques, one can stumble upon modelling of interface, content, or structure. Interface modelling falls under presentation modelling, and content falls under information modelling. Structure modelling usually is about information modelling, but it can talk about presentation or navigation models as well. It is just a different name, and not a different kind of technique. Other than that, all the design approaches found in the table will be discussed in the thesis with respect to requirements processing.

## 3.2. ABSTRACTION LAYERS IN WEB INFORMATION SYSTEM MODELLING

The idea that is provided by [11] is to use an Abstraction Layer Model for Web-based systems. It is not exactly a modelling technique per se, instead, it is like a tool to put relevant information into relevant places.

There are five layers distinguished, where all of them except the first are additionally defined to have two dimensions: *focus* and *modus*. The idea of focus is to distinguish local and global components; modus distinguishes between static and dynamic components. This leads to four combinations. The distinguishing between components is summed up in the table 3.2 below. It says which specifications address which components. Just to make a note, components are referred to be certain models of presented types of techniques.

**Table 3.2.** ***Dimensions of modelling techniques***

| Modus\Focus | Global | Local |
|---|---|---|
| **Static** | Data specification | View specification |
| **Dynamic** | Function specification | Dialogue specification |

Every of the requirements processing techniques falls under one of these four categories. We will come back to this in the taxonomy after all the solutions will be presented.

The layers that [11] gives are these:

Layer 1: *strategic layer*. This layer describes the system in a general way. This basically corresponds to a mission statement. (Mission statement in more detail is discussed in 4.3) The transition to the next layer is associated with activities of story boarding and user profiling. Also discussed in 4.3.

Layer 2: *business layer*. This layer is to identify more specifically kinds of users and their profiles, roles of users and tasks associated with these roles. This layer also deals with storyboards: paths through the system and information. The transition to the next layer is associated with conceptual modelling (that includes database modelling, operations modelling, view modelling, and media type modelling). This is the majority of what the part 4 of this thesis talks in general.

Layer 3: *conceptual layer*. This layer is all about analysis and integration of scenes taken from story boards. Each scene has to be supported both by content and functionality. The transition to the next layer is associated with definition of presentation styles. Given in part 4.5.

Layer 4: *presentation layer*. It is concerned with associating presentation options to the media types. Transition to the next layer is associated with all implementation tasks.

Layer 5: *implementation layer*. This layer is responsible for implementation: setting up logical and physical databases, page layouts, realisation of functionality, etc.

So it would be more visual, the following figure 3.3 illustrates the layers and transitions between them.

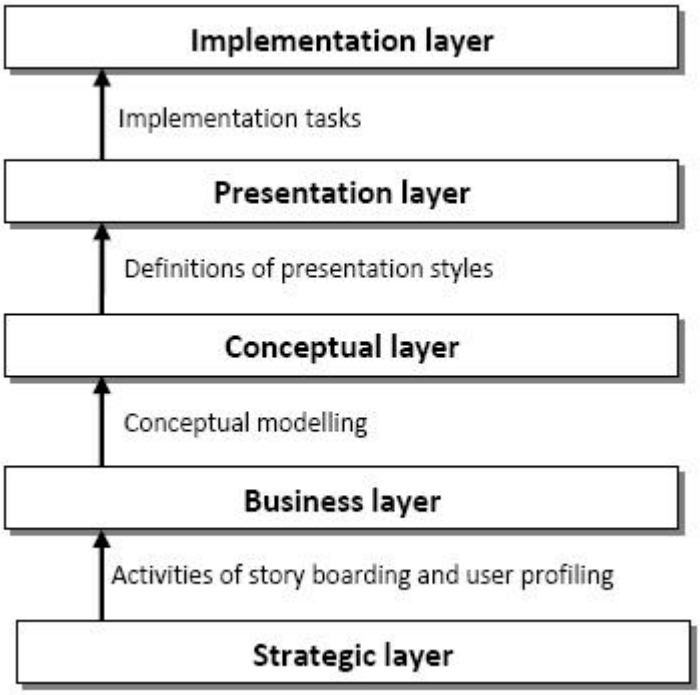



**Figure 3.3.** ***Layer transitions***

This way of putting a Web design processing in order is what [11] suggests. This is more like an example of how things can be done but usually designers have to choose individual way of

making the work process. Nevertheless, this is an interesting example here to us because this thesis talks about requirements processing with regards to all of the layers except the last one (implementation layer). In the last part of the thesis it is shown how the presented techniques fit to this solution. But, again, it is just one example out of many possible of how a designer can formalize a development process of Web based information systems.

# 4. REQUIREMENTS: FROM STATEMENTS TO MODELLING COMPOSITIONS

This part of the thesis introduces with five different aspects of requirements processing. We have already discussed the fact that modelling techniques are very important in visualizing requirements which leads the stated requirements becoming the real requirements, and in the end user requirements becoming software specifications for that matter. Aspects of information, navigation, operation, behaviour, and presentation are here discussed with respect to Web based information systems.

## 4.1. INFORMATION MODELLING

As it was already established, information modelling is one of the most popular design techniques in Web based systems. The overall idea is rather simple – the result is all about specifying information items. This usually ends up in some conceptual schema, which generally is a UML class diagram; that is indicated at least in both: Güell, N., Schwabe, D., and Vilain *Modeling Interactions and Navigation in Web Applications* [1] and N. Koch, A. Kraus, C. Cachero and S. Melia *Modeling Web Business Processes with OO-H and UWE* [2].

The hard question is how to manage given requirements correctly in order to achieve the desired and accurate result. This part analyses two different approaches for this matter. The first approach given in [1] puts all the requirements into user interaction diagrams (UIDs) and then, by applying predetermined rules, forms a class diagram. The second approach given in [2] prefers use case diagrams (UCDs) for the representation of functional requirements.

An interesting conflict here is that [1] argues that UML does not have a diagram that would be equivalent to the idea of UID. While the argument seems to be valid, the question is about picking the right technique. UID is able to capture navigation and interaction details, whereas UCD captures just strictly certain user-system interaction aspects. The first approach is better whenever the user provides really detailed requirements and the system is possibly complex. The second approach fits more if the user knows what has to be done, without having any idea of how that can be achieved.

### UID BASED APPROACH

The scheme for requirement handling for simple Web based information systems is nicely constructed in [1]. The idea the authors present is all about walking a way of requirements gathering, making a specification of conceptual schema and directly building one.

The requirements gathering from users and all the relevant stakeholders have these strictly determined steps: identification of roles and tasks, specification of scenarios, specification of use cases, specification of user interaction diagrams, and validation of use case and user interaction diagrams. Figure 4.1 illustrates the sequence visually.

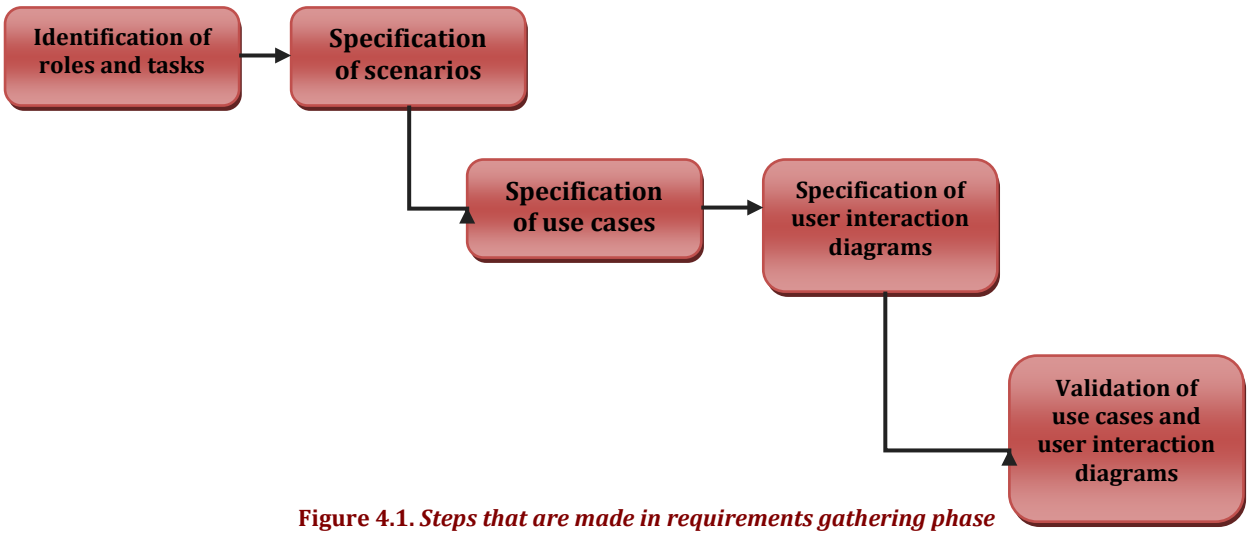



**Figure 4.1.** ***Steps that are made in requirements gathering phase***

*Step 1: Identification of roles and tasks.*

The goal of this step is to exactly identify what kind of roles users might play with regards to the system under development and, what is more, what kind of tasks it will have to support. In this way we map the possible roles to actual tasks. This step is a starting point of figuring out functional requirements, so the main focus has to be held on the functional side of this matter. The process at least has to involve interviews with relevant stakeholders. Other than that, it might also require analysis of prepared documentation.

As an example, a simple Web system *www.nfit.au.dk* will be analyzed. The description of this system can be found in appendix B. Primarily three types of roles of users can be distinguished: a role of a student, of a teacher, and a role of an administrator. Now tasks, then again, can be as simple as to follow the link in order to access a personal Web mail account; or more complicated as accessing networking information. The representation is given in figure 4.2.

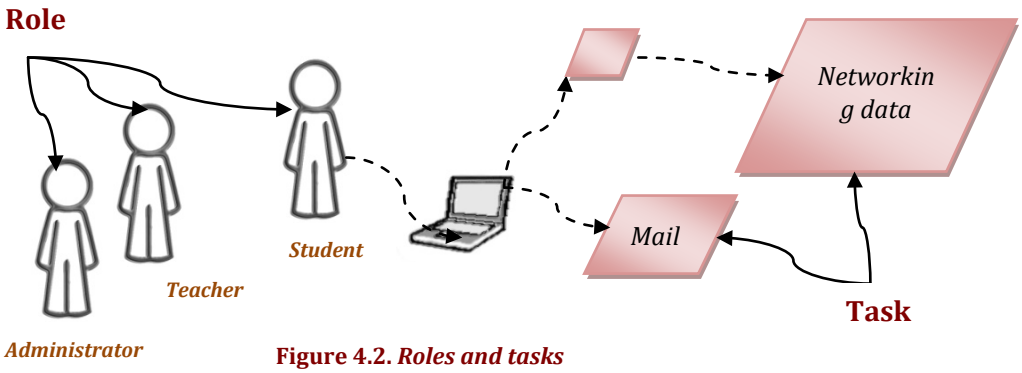



**Figure 4.2.** ***Roles and tasks***

*Step 2: Scenario specifications*

The goal of this step is to gather up as many relevant scenarios as it seems to be relevant due to the system design. For that, all the relevant stakeholders should participate in providing scenarios in a written or spoken way. Although it is suggested to concentrate mostly on system users, to include other stakeholders would not hurt. It would only benefit in making scenario specifications more complete.

Scenarios are descriptions of how the system may be used. With regards to users, scenarios for each kind of roles have to be specified. The roles were identified in step 1. What is more, tasks that were collected have to serve as guidelines while determining relevant scenarios.

Let's consider a simple example. Previously a task was identified by a user which had a student role: accessing Web mail. A detailed scenario that goes with this task would be: opening the main page, following the relevant link 'Web Mail'; now the system should open relevant page where student's user name and password is asked; after inserting relevant credentials, the system should open student's Web mail account; see the figure 4.3.


**Figure 4.3.** ***An illustration of one out of many more possible scenarios***

*Step 3: Use Case specification*

The goal of the use case specification is very simple: to group all the scenarios describing the same task together and give a description of that with the most relevant information. If users representing different roles presented scenarios about the same task, all the roles have to be mentioned explicitly. An example of a use case description is given in the table 4.1 below.

**Table 4.1.** ***An example of a use case***

| **Use case** | **Accessing Web mail account** |
|---|---|
| **Scenarios** | Scenario 4; Scenario 17 |
| **Roles** | Student; Teacher |
| **Description** | 1. The user goes to the main page of the system where one finds a link to the Web mail system; the link provides with the Web mail systems page where user name and a password is asked; after providing the right credentials the user is given an access to the account<br>2. The user goes to the main page of the system where one decides to first review the information about the mail server, so one follows 'Engineering and dock.' link, finds 'Mail system' link, then 'NFIT mail' link where one can read some information about the mail system and then go and access personal mail account by following 'Webmail' link; the link provides with the Web mail systems page where the user's name and password is asked; after providing the right credentials the user is given an access to the account |

*Step 4. User Interaction Diagram Specification*

The idea of this step as [1] suggests is to show interactions between the user and the system without considering any of possible interface aspects. For that user interaction diagrams (UIDs) are used. Interestingly UML is not considered as a proper mean for these kinds of diagrams and the argument for not using is that no certain objects are identified yet. Nevertheless, there are opinions, like in [2], that indeed UML's activity diagrams are the most expressive tool in properly defining use cases. However, these issues will be analysed more deeply a bit later. Anyhow, instead of UML, [1] gives gadgets as ellipses, arrows, and lines. See the figure 4.4 below.

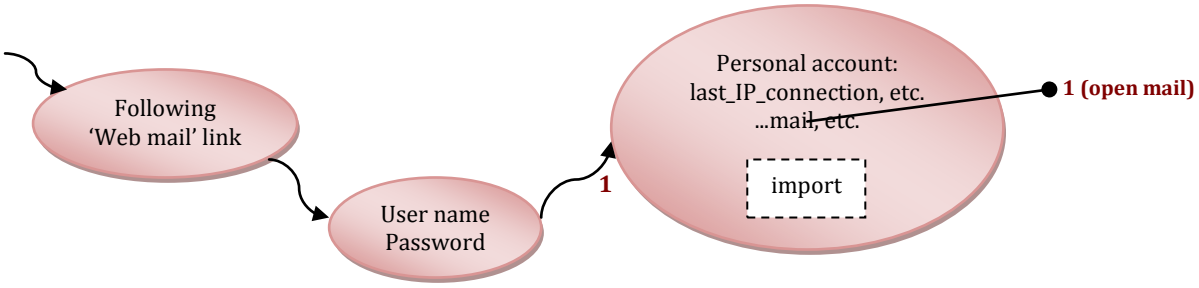



**Figure 4.4.** ***User Interaction Diagram of 'Accessing Web mail account'***

This is an example of how a use case can be mapped to the suggested way of making a user interaction diagram. The user in the main page of the system follows the link 'Web mail'. The system gives a page where a user name and a password is required (this is an only choice of the input). The system then gives a page with a personal mail account together with loads of information. Part of that information is an exhibited attribute 'last_IP_connection' indicating an IP address from which the account was accessed last time; an exhibited attribute '...mail' indicating a list of e-mails, whereas each of them is associated with 'open' operation. As an example of an optional input here is an 'import' attribute which allows a possibility of importing an address book.

In this example 'open' operation is the one that is associated with an information 'object', e-mail that is. As it was given in [3], and it was discussed a bit earlier, operations can be associated with links (navigational structures) as well.

The exact meanings of all the elements are listed in this table 4.2 below:

**Table 4.2.** ***UID elements***

| Element | Meaning |
|---|---|
| **An ellipse** | An interaction between the user and the Web system |
| **An arrow, connecting ellipses** | Some processing in the application occurs before some other information is presented |
| **An arrow without a source** | The first interaction |
| **A line with a black bullet in the end** | Operations that do not require the interaction exchange |
| **Ellipse filling** | The inside of an ellipse is filled either with user's input or with system's output |
| **Triple dot** | The triple dot in front of an exhibited attribute means that there is more than one element (a set of them; it is possible |

| | |
|---|---|
| | for the set to be empty) |
| **A number** | A number close to an arrow or to a line representing an operation indicates number of choices |
| **Dashed square** | Indicates an optional input |

One more approach, similar to UIDs, is proposed by [11]. Instead of gathering many different user interaction diagrams, an idea of storyboarding could be used. It is all about an activity that addresses the design of an underlying application story. This is mostly used in more complicated systems and is somewhat beyond requirements information processing, therefore, this approach is discussed a little bit further.

*Step 5. Use Case Validation*

The goal of this step is an explicit communication with users and all the relevant stakeholders about collected use cases and UIDs. The main thing is to discuss all issues with associated roles. All representatives of the same role have to reach a consensus about related use cases and UIDs. With regards to that, some documentation has to be made in order to keep track possible alternatives.

After these five steps have been made, we have all the roles for the system identified and for those roles there are all possible and relevant use cases found and user interaction diagrams made. The amount of information possessed is now quite big. What is import at this point is to use it correctly.

What [1] suggests is a set of strict rules for specification writing out of which a conceptual schema (or a class diagram if to be precise) can be modelled. As not too much interpretation about these rules can be made, these will go here as they are given in [1].

1) This rule takes into account every UID. Taking each of them, classes have to be defined for each element, set element and specific element. For each class that is defined, an identifying attribute has to be assumed.
2) This rule deals with defining attributes. These have to be defined for each data item of the set elements that appear in each UID and for each data item imputed by a user. While defining, one has to be aware if:
    - It can be verified that the data item is functionally dependent* on the identifier in each class; if it can be verified that the data item is not transitively dependent** on the identifier. The data item that satisfies these conditions becomes an attribute.
    - It can be verified that the data item is functionally dependent, but not transitively dependent on the identifiers of two or more different classes and that the data item is not transitively dependent on the identifiers. The data item that satisfies these conditions becomes an attribute of the relationship between these classes.
3) Attributes have to be defined for the data item that is entered by the user but represented by a single rectangle. Two complementary conditions that were applied in 2) rule have to be taken into account here as well.

4) This rule deals with relationships between classes. Each attribute has to be taken one by one. If an attribute appears in a set, different from its class, a tentative relationship has to be defined between its class and the class of the set elements. In case they are not related, it has to be verified that the attribute class is related to a class of another attribute presented in the set. Nevertheless, all the relationships have to be verified to be semantically correct.
5) For each interaction in the UIDs, if the source class differs from the target class, a relationship between these classes has to be defined. It has to be verified that defined relationship is semantically correct.
6) It has to be verified that each operation in UIDs corresponds to the relevant class.
7) Further necessary adjustments have to be done, like, possible generalizations found, missed relationship cardinalities defined, etc.

For those that are not sure what exactly dependencies are, here it is some explanation:

*functional dependency is basically a constraint between two sets of attributes in a form of a relation.

**transitive dependency is a relation between three or more attributes: if A→B, and not B→A, and B→C, then A→C is a transitive dependency.

This set of rules would seem to be descriptive enough but, nevertheless, in every step the one responsible for the schema building has to use ones skills and intuition, especially when verifying semantical aspects. With respect to the example that has been analyzed all the way through so far, a mini example (one of possible versions) of a conceptual schema can be seen in figure 4.5.

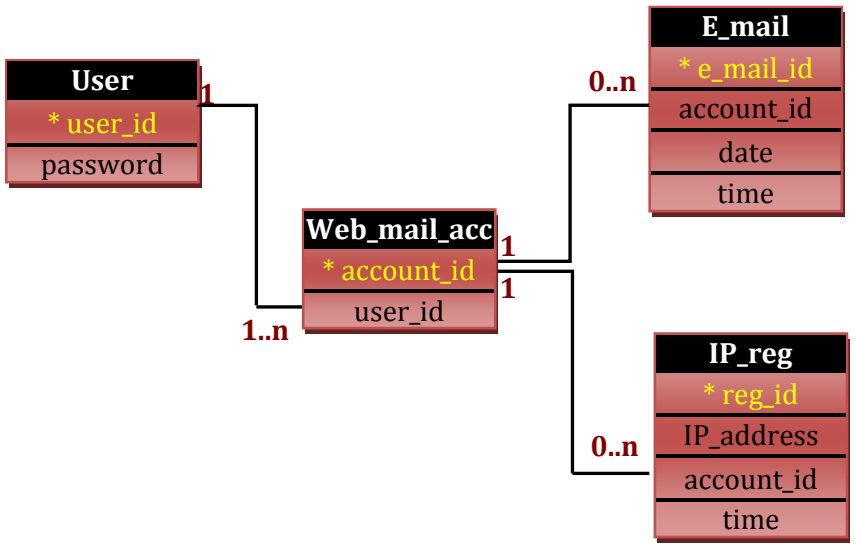



**Figure 4.5.** ***An example of a conceptual schema***

To build this conceptual schema, we take into consideration only the UID that is found in the figure 4.4. We stick to the main four elements that can be found in that diagram: a user (an account has an owner, and it has to have only one owner for that matter), a Web mail account (if a user is registered in the system, one has at least one account), an e-mail (there can be many e-mails in the account, or at some cases possibly none), and some IP registry (account's every access is registered).

A user is identified by user's identification number and, for a matter of connection, only one's password is relevant as a defining attribute (it is an input of a user). Every user can have one or more Web mail accounts, whereas the account is identified by some identification number and, so it could be matched with a particular user, user's id has to be a defining attribute. What is more, there can be none or more e-mails in a certain mail account. Each e-mail has its identifications and lots of information that go with it (date, time, etc.). Subsequently, for a particular e-mail to be matched with a particular Web mail account, account's id has to be a defining attribute. Analogous situation is with the IP registry. It is worth mentioning, that if, for example, we would like to take user's id as a defining attribute in the e-mail class, then it would satisfy the transitive dependency which would violate the rules defined, therefore, it is not a defining attribute there.

As it was mentioned, it is just a miniature illustration of a transition from UID to a conceptual schema. But, at the end, they both matter. Conceptual schema reveals more information pieces and relations between them, whereas a user interaction diagram captures some navigational aspects as well.

## USE-CASE BASED APPROACH

Before going into the details of this particular approach, there are some aspects that can be borrowed from the previous part (from UID approach). That is *Step 1* to *Step 3*. This is because [2] does not suggest other technique for that matter, and the one established in [1] is pretty common. The flow of steps for this approach is illustrated in the following figure 4.6.

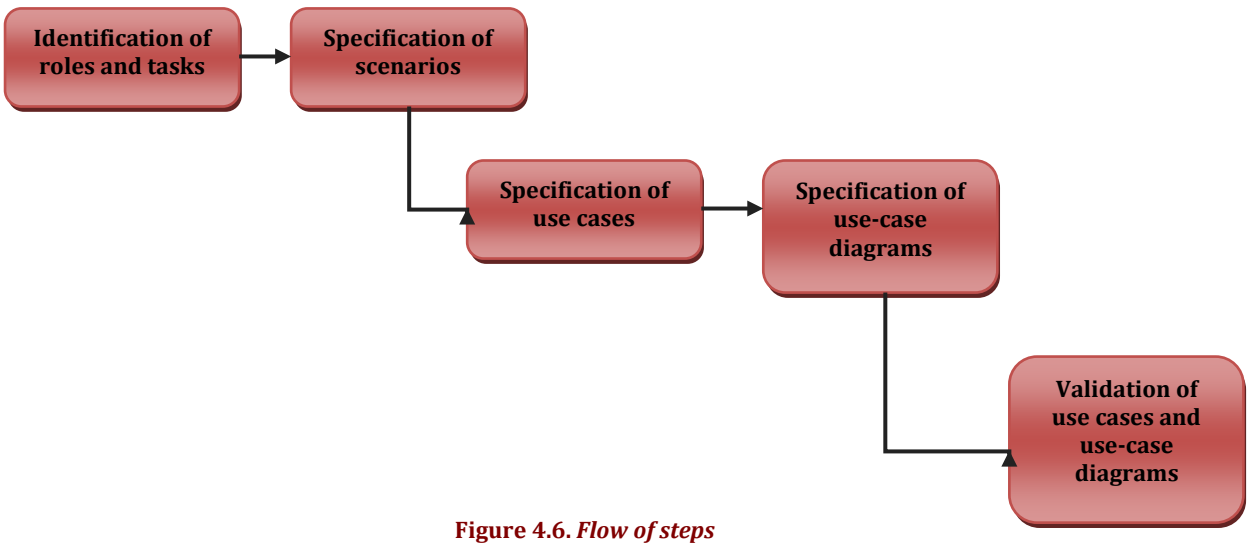



**Figure 4.6.** ***Flow of steps***

As it can be seen, we do the same as before just deal with use-case diagrams instead of user interaction diagrams. As a result, we will pursue with the same example of a simple Web system of *www.nfit.au.dk*. By doing this, it will be easier to compare these different techniques given by [1] and [2].

*Step 4. Use-Case Diagram Specification*

The idea of this step as [2] suggests is to show interactions between the user and the system without considering anything else but functions that the system delivers to the user. And, as [2] does not provide information of how to build a use-case diagram, we will borrow some working ideas from [16] and incorporate them with ideas from [2].

This is the way of working that [16] suggests, having in mind that we have already identified context of the system, actors, and defined use cases in previous steps:

- Transfer ideas of the common behaviour into new use cases that are used by others; transfer variant behaviour into new use cases that extend more main line flows.
- Model these use cases, actors, and their relationships in a use case diagram.
- One can complement these use cases with notes or constraints that assert non-functional requirements.

What is more, boundaries of the system have to be strictly known, that is which behaviours belong to the system and which are performed as external entities. Similar actors can be organized in a generalization-specialization hierarchy, and where relevant, stereotypes can be used.

The following example (figure 4.7) tries to illustrate the part of the system which was used in UID in the previous part of the thesis (figure 4.4). It is all about accessing the mail system, signing in, and getting the mail.

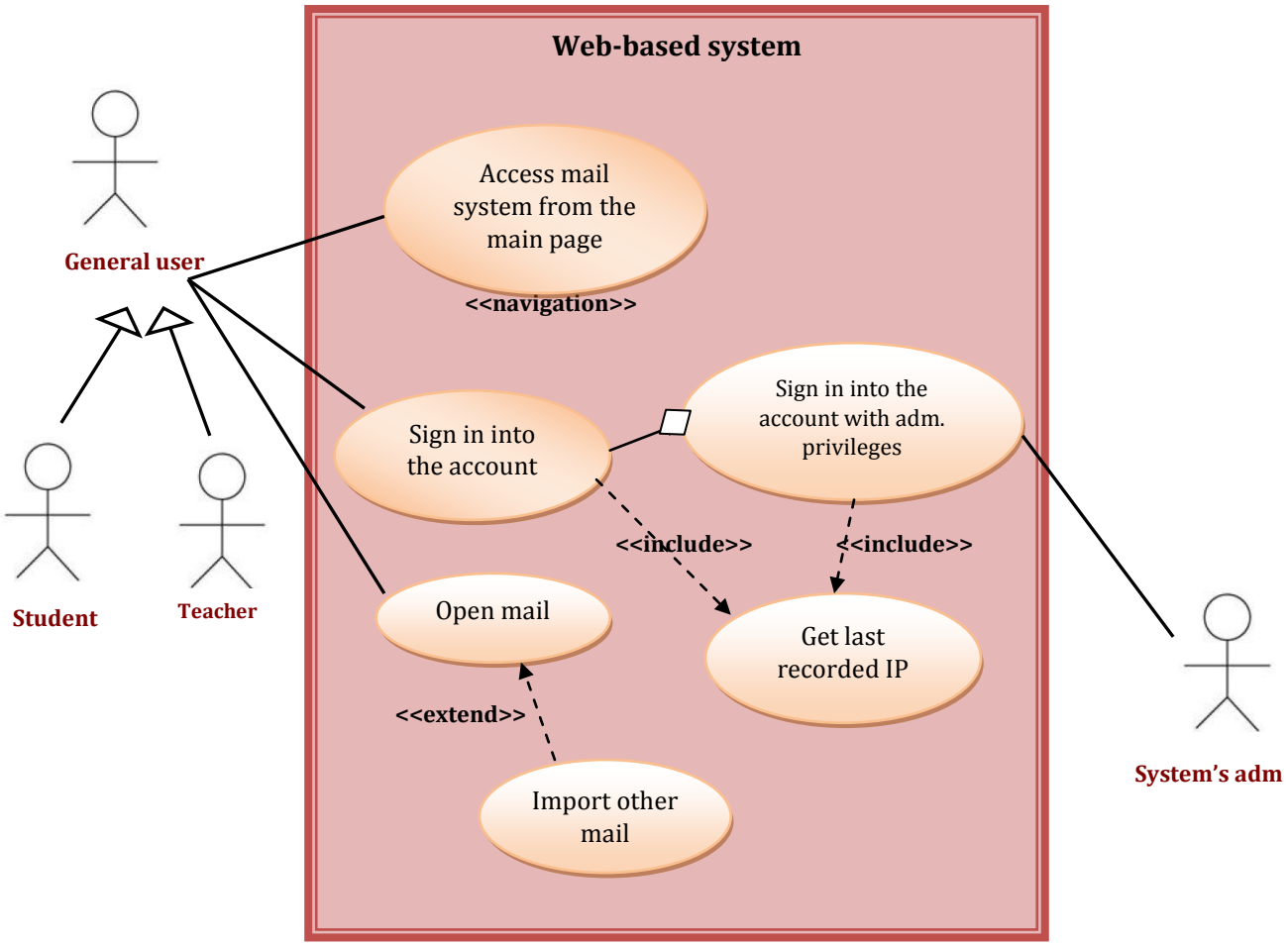



**Figure 4.7.** ***Use-case diagram***

In this use-case diagram we have two types of users (actors): general user and the system's administrator. General user performs the roles of a student and a teacher, whereas system's administrator performs administrator's role. General user can access the mail system through the main page (this use case acts only as a navigation unit), what is more, one can sign in to the mail account, and get all the e-mails.

This diagram indicates that 'student' and 'teacher' are specific roles of the generalized 'general user'. Now 'sign in into the account' has fewer functions than that with administrator's privileges, which means that it is a part of a whole, therefore aggregation relationship is used.

What is more, it is implicit that after signing in into the account, one gets an IP address indicating where from the last connection was made. Other than that, for use cases that may be performed explicitly 'extend' stereotype can be used indicating that.

Overall, when building use-case diagrams for a system, which is not totally simple, there isn't a right way to do that. There are many levels of functionality that can be captured; and the perception of the relationships among units of use cases is not trivial, having in mind that it captures the requirements and all the assumptions about the implementation of the system. For that reason, a use case diagram should capture the most relevant aspects of functionality. What is more, it has to make sense not only to the designer but to all the relevant stakeholders as well. The table 4.3 below summarizes the elements of the use-case diagram.

**Table 4.3.** ***Elements of a use-case diagram***

| Element | Meaning |
|---|---|
| **A rectangle** | The rectangle indicates the boundary of the system, and it is called *subject boundary* [16]. On the top part of the rectangle, the name of the system has to be written, so called *subject*. |
| **An ellipse** | An ellipse describes a specific unit of a use case. |
| **A little man** | A picture of a little man indicates an *actor*. |
| **A line** | An ordinary line indicates that an actor can use the system as it is described in that specific use case. |
| **An empty arrow** | An empty arrow indicates a generalization relationship. |
| **A diamond arrow** | A diamond arrow indicates an aggregation relationship. |
| **A dashed arrow** | A dashed arrow indicates somewhat dependency between two use cases. |
| **<<?????>>** | This is called a *stereotype*.<br><<include>> - a use case is performed implicitly;<br><<extend>> - a use case is performed explicitly;<br><<navigation>> - a use case that expresses navigation activities; this stereotype is used in [2], so it would be possible to capture navigation requirements in the use-case diagram, although it is not a typical way to use use-case diagrams. |

If to compare UIDs with use-case diagrams, one can see some commonalities: both diagrams can capture some sense of navigation and give the idea of what the system can do. The advantage of the UID is that it is able to show specific units of information in the interaction process, whereas the advantage of the use-case diagram is that it can give relationships among the users and use cases and within the use cases in general.

*Step 5. Use Case Validation*

The goal of this step, as before, is an explicit communication with users and all the relevant stakeholders about collected use cases and use-case diagrams. The main thing is to discuss all the issues associated with different situations in the system. Usually, there are more than one use-case diagram for a system (although there could be one) but normally that could be too confusing to understand. The refinement can be done with respect to different roles of actors or with respect to different levels of functionality, whatever suits stakeholders the best.

After all the steps are done, all the requirements that were agreed on have to be put into some conceptual schema. And this time, there are no such strict rules as before. What is more, [2] does not suggest any new ideas for doing this, except that it is all about UML class diagrams. For that matter, we will borrow some basic rules from [16].

- One has to identify those classes whose state must remain the lifetime of the system.
- Create a class diagram that would contain these classes. Sometimes it is a good idea to define an individual set of stereotypes and tagged values that would address future database-specific details.

- For all the classes, details of the attributes have to be defined, associations established, focusing on the multiplicities that relate relevant classes.
- One has to watch out for cyclic and one-to-one associations. Sometimes it can be beneficiary to create intermediate abstractions in order to make the logic of the structure less complicated.
- One has not to forget that during the design process lists of attributes and operations tend to grow.

In the following figure 4.8, there is a conceptual schema corresponding to the modelled use-case diagram (see figure 4.7). It is a simple schema, providing just the most relevant pieces of information, and giving examples of somewhat interesting relationships among the classes. The idea that [2] originally gives is separation of 'what belongs to the system' and 'what belongs to the user'. We have two packages here, the first one is all about structures maintaining information related to the domain objects and the second one takes into account structures related to the individual user. The given example might not be the best one, but it still gives the right idea of this approach.

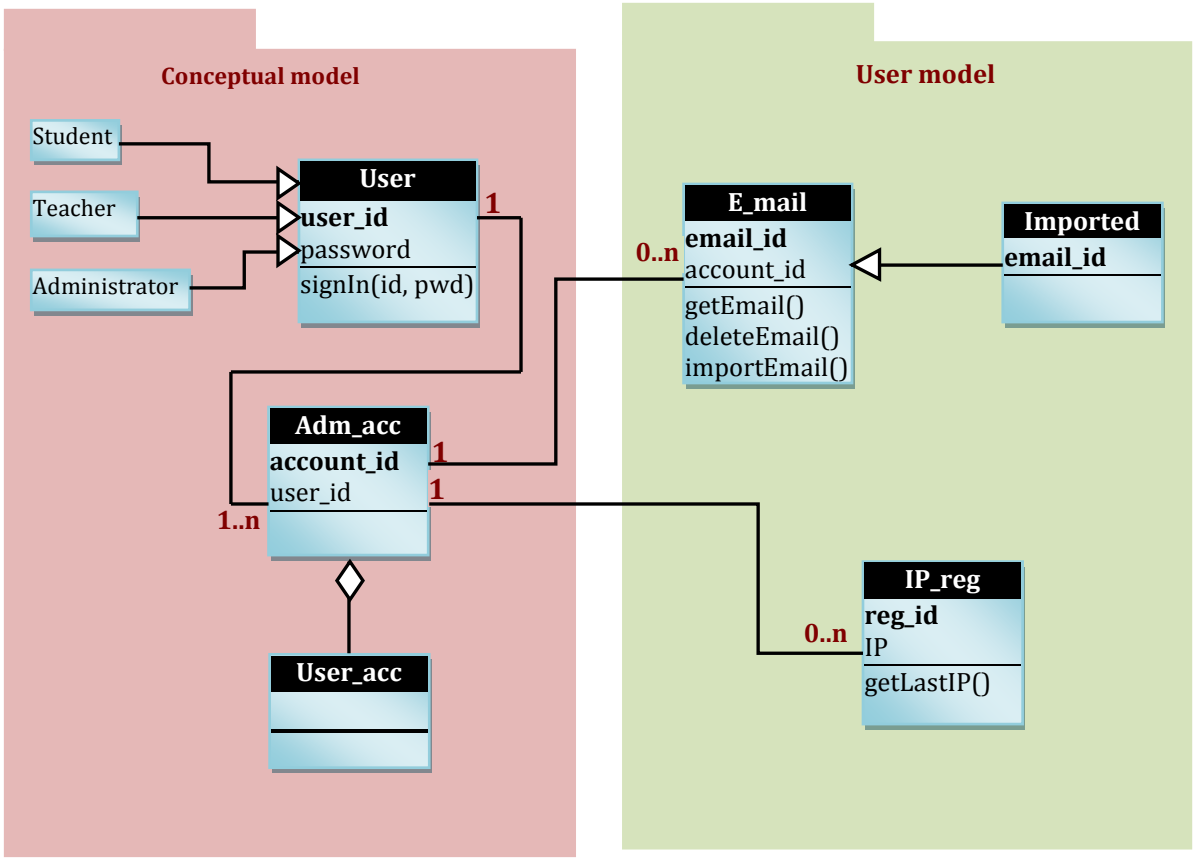



**Figure 4.8.** ***A conceptual schema of a use-case diagram***

The given example has two packages – a conceptual model (information about the domain objects) and a user model (information concerning the individual user). In this case, on the one hand, the domain side is concerned with mail accounts and people using those accounts, on the other hand, e-mails and IP registry is more relevant to the individual user.

What we get from the use-case diagrams is the idea of objects one will need information about during the whole lifetime of the system; that is: a user, an account, an e-mail, and IP registry. One more thing that we can get from the use-case diagram is some relevant relationships. It is often the case that one piece of a use case functionality is implemented in a single class. We will use that tactics here. But first of all, one has to understand that there is no direct translation from a use-case diagram to a class diagram. It is up for interpretation. The most important thing is to capture all relevant requirements with the former one and the requirements have to be well understood and agreed on by all the stakeholders. In the mean time, conceptual model has to provide requirements in the light of implementation, possibly with somewhat different solutions for the problems, but, at the same time, fulfilling all the relevant functionality.

Following these thoughts, we consider that a student, a teacher, and an administrator are all of types 'User' and in order to show that - we use generalization relationship. In the use-case only a student and a teacher were specializations of a 'General user', but describing all the users being of the same type gives more rationale here.

The use-case diagram gives the idea that there are two sets of information that an account class may provide: for an ordinary user and for the administrator. The 'Adm account' (giving relevant rights for the administrator) absorbs more functionality, therefore, it aggregates the 'User account'. What is more, we can import e-mails from elsewhere and the fact that these are of type 'E-mail' are indicated by a generalization relationship.

If to make a little comparison with the former conceptual schema that is in figure 4.5, we can see that it is more complete in a sense of information units. This is because UIDs can better capture all the aspects of information that are used in the system; therefore, to build classes with attributes are a bit more straightforward. The conceptual schema that we just analyzed here is better in showing right relationships between the classes (as a class corresponds to the use case unit) but not the attributes that the class should contain. That part is left to figure out for the designers.

## DISCUSSION

In this part, which deals with information modelling in Web systems, we analyzed two different solutions. The first approach is based on putting the requirements into user interaction diagrams (UIDs) and then transforming those into conceptual models (class diagrams). The second approach, alternatively, puts all the requirements into use-case diagrams and then these diagrams are used to model conceptual schemas.

While both approaches are seeking for the same goal, the result differs a little. The first approach results in a conceptual schema which contains all the relevant classes; and all the attributes for those classes are directly picked from UIDs. The hard work to be done is putting the right relationships between relevant classes. The second approach gives conceptual schema with the relevant classes as well, but this time relationships can be handled here more accurately. The hard work at this point is to determine the right attributes.

## 4.2. NAVIGATION MODELLING

Requirements putting into navigation modelling structures are one of the most important activities in requirements processing that can be done working with Web based information systems. And navigation should be understood not only as going from one page to another. There are lots more to that and all the relevant ideas are given in this part of the thesis.

There are five solutions presented of how one can deal with requirements processing with regards to having an outcome of a navigation model. The first approach is UID based which is presented in [1]. In the part before, we had some information modelling ideas that were based on this type of diagrams. The navigation part can be seen as a continuation to that. The second and the third proposal are based on building so called NADs (navigation access diagrams). These are based on thoughts given by [2], [4], and Cachero, C., Gómez, J., and Pastor, O. *Object-Oriented Conceptual Modeling of Web Application Interfaces: the OO-H Method Abstract Presentation Model*[17]. NADs are based on the idea taken from object-orient hypermedia approach but interpretations of that can differ. The fourth of the ideas talks about building navigation models with respect to data (information) items and functional items. Proposals are given by [4] and Quintero, R., Torres, V., Ruiz, M., and Pelechano, V. *A conceptual modeling approach for the design of web applications based on services*[23] but the given solutions are different, so they are discussed separately.

### UID BASED APPROACH

This approach precedes the idea of user interaction diagrams that was proposed in [1]. Just to remind, UIDs were analyzed in part 4.1 while presenting the approach of information modelling. In this part we will see how to process requirements in order to get some navigational structures. The most important inputs for this matter are scenarios provided by stakeholders, use cases formed out of those scenarios, and UIDs. The latter set can be objectively verified by relevant stakeholders so it can be considered as a really trustable foundation of information. That is, of course, until the next stage of verification.

The first thing to do in order to get a full navigational map for a system is to create a navigational structure for each known task. That is, for each task to determine the most appropriate navigation sequence that supports the user.

For this matter, [1] suggests to use a concept of a 'navigational context'. The definition given is: 'the navigational structure of an application is characterized as a group of contexts in which the objects will be accessed'. Navigational context is presented in a form of a context diagram. So it would be easier to follow how everything works, the next figure 4.9 illustrates an example of a context diagram based on the running example about accessing the mail account, somewhat corresponding to the UID given in Figure 4.4.

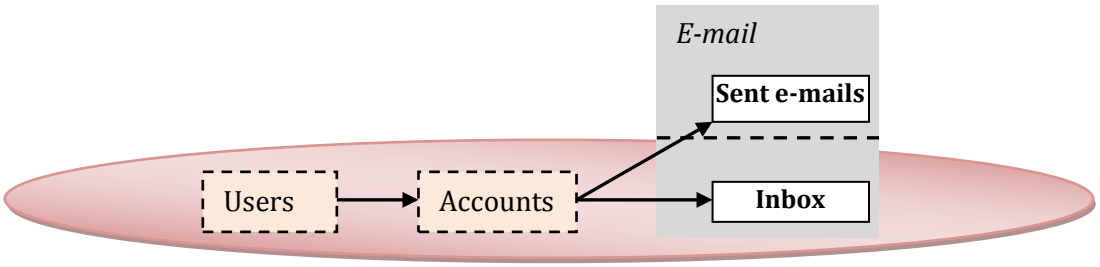



**Figure 4.9.** ***A context diagram***

The overall idea of the context diagram is that a user is determined by a user id (or log-in name); the right credentials match to the relevant account. The most interesting aspect is met with the e-mail. The object of an e-mail can be found within two contexts: in the context of the original inbox and in the context of those e-mails that were sent. The table below summarizes all the elements of a context diagram.

**Table 4.4.** ***Elements of a context diagram***

| Element | Meaning |
|---|---|
| **An ellipse** | An ellipse indicates boundaries within which the solution of the given UID is resolved ('sent e-mails' is out of boundaries because the analyzed UID originally did not contain any information about those). |
| **A dashed rectangle** | A dashed rectangle specifies an access structure, which will be discussed later. |
| **An arrow** | An arrow shows navigational relationships. |
| **A grey-filled rectangle** | A grey-filled rectangle presents an object or a class (in the example it is an 'e-mail'). |
| **A non-dashed rectangle** | A non-dashed rectangle indicates contexts within which the class or the object may exist. |
| **A dashed line** | A dashed line separates different contexts of the class or the object. |

Here we analyze just a simple example in order to get familiar with symbols that we need to build a context diagram. As a matter of fact, [1] proposes some rules that help to build these diagrams quite accurately. This is based on the idea that in a system all the units of information can be grouped into sets. This is how we get objects and classes that are represented in conceptual models. Now, as far as context diagrams go, each set found in UID have to be analyzed and typed. Finally, a set becomes an access structure, a context, or a list attribute. The mapping rules, suggested by [1], that have to be followed while analyzing UIDs are summarized here:

- Mapping of the interaction that presents an object, into a context of the object class;
- Mapping of a set that is part of the information of an object into an access structure, or into a context, or into a list attribute;
- Mapping of a data entry interaction followed by an interaction that presents a set of objects, into an access structure of the set's objects.

Before going into the detailed analysis of these rules, we will take a new example of a UID (figure 4.10), which would be able to provide as much information as possible with respect to the amount of useful information this technique can give.

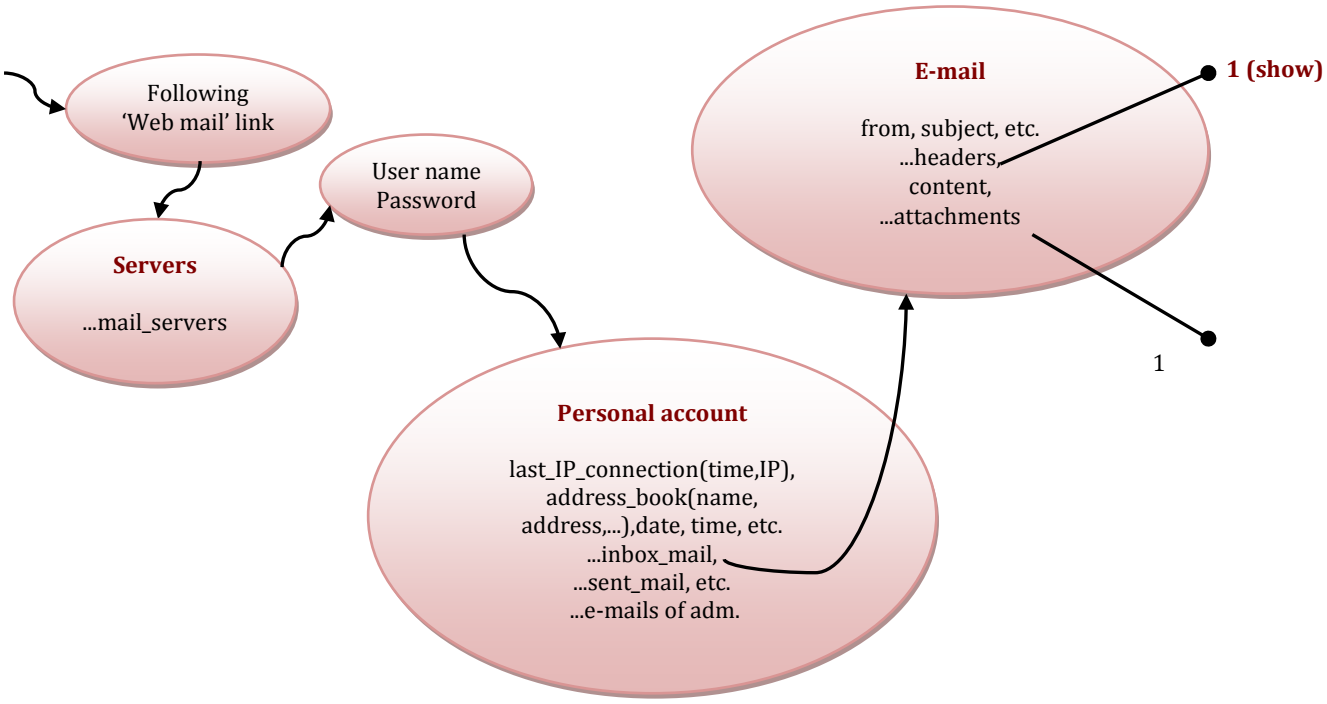



**Figure 4.10.** ***An example of UID***

The idea of UID building was explained in the part 4.1 before, so this is no longer an issue to talk about. Nevertheless, it is reasonable to share some thoughts of what this particular UID gives to us. Now, as before, we follow a 'Web mail' link. But in this case we assume that the system allows us to choose more than one mail server to connect to the account. After choosing the relevant server, we have to provide right credentials. This gives us the access to the personal account which provides loads of relevant information. There is one interesting aspect, though. In order to access an e-mail, we have to choose that from the list. For this matter, the arrow goes not from interaction to interaction (not from an ellipse to an ellipse) but from an element indicating the list to the unit of interaction. This is because the right e-mail is the result of us selecting it from a list. Now, when we are in the e-mail, it provides all the relevant information that e-mails usually do.

Now it is time to analyze this UID according to the rules we defined before. With regards to this, the first rule suggests to map object presenting interactions into a context of the object class. In the UID we can see that we have three object presenting interactions: 'Servers', 'Personal account', and 'E-mail'. At this point, the thing to do is to determine the elements of each context. These elements, as given by [1], have some common characteristics: they belong to the same class, they have attributes with the same value, or they are linked to a given object trough the same relationship.

In order to determine elements of context, sources of interactions in the UID have to be determined. There are some guidelines for that:

- In case the source of interaction is within a set of objects that have a common characteristic, the natural elements of the target context are these same objects.
- In case the source of interaction is another object, the elements of the context are those that are linked to the source object trough a certain relationship.
- If it is an initial interaction, it is not possible to specify the objects of the context. In this case [1] suggests to adopt the convention of labelling these contexts with '<Class name> by ?'.

Having in mind these guidelines we can see that the interaction that presents the object of class 'E-mail' is accessed from the interaction that presents the set of all the e-mails in the account. Because of this reason, the object of this context will be that user's account, which contains these e-mails. Now the interaction that presents the object of class 'Account' is accessed through the credentials providing interaction, which is accessed from the server selecting interaction. This sequence can be described by a relationship, as a certain server contains a certain account (or a certain account belongs to a certain server). For this matter, the object of this context will be the corresponding mail server.

The following figure illustrates all the thoughts given above. It is just worth mentioning, that this example is not too complicated in a sense that an e-mail belongs to one account and that one account is on one mail server. The idea of context object becomes more than very much relevant if e-mails can be shared among accounts, and accounts among servers. Nevertheless, it is not the place to analyze complicated situations, and the figure 4.11 gives just the right taste of this idea.

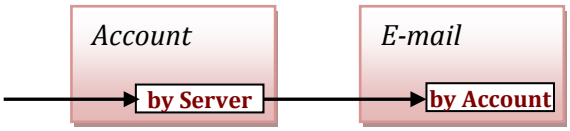



**Figure 4.11.** ***An example of context objects***

The second rule suggests mapping a set that is a part of the information of an object, into an access structure, or into a context, or into a list attribute. And as we will see, this is the place where informational aspect of the requirements gets categorized. It is interesting that it is done in navigation modelling part but this is where it becomes relevant. Nevertheless, it is impossible to absolutely separate different design approaches.

In the requirements it is stated what kind of information a user expects to get displayed with regards to the server, to the account, and to the e-mail. If it is a separate piece of information then it might be straightforward to place it where one could see it. Now, if it is a piece of information that is in a set or some data providing lists have to be made, there is no way that everything can be displayed in one place. Following this thought, sets of information have to be categorized by importance for a user immediately to see it. There are three categories: very important, important, and complementary. And although [1] suggests this idea only dealing with sets of information, the same idea works perfectly well with separate units of information too.

In the UID example (see figure 4.10) we have that interaction 'Account' has sets as inbox mail, sent mail, and contacts of administrators. And interaction 'E-mail' has sets as headers and attachments.

Now here are some thoughts about mapping. What that rule talks about is first of all deciding on the importance of information with respect to the execution of the tasks that will use a particular object (for this matter, objects of 'Account' and 'E-mail'). At this point it is not enough just to analyze all the relevant UIDs. One has to go back to the foundations of the requirements – the scenarios and adequate use cases. The idea here is to very well understand tasks and users' goals. For this particular problem some other techniques can be used, such as task-goal trees

which are capable to capture the aspect of importance very well. This technique will be presented in the later parts of this thesis.

Paper [1] suggests dealing with different categories of information in this way:

- Very important information should appear integrally as lists that are attributes of the object;
- Important information may appear as access structures tied to the object. In this case the designer should decide the attributes that will be presented in the structure;
- Complementary information can appear as an anchor that points to an access structure or to an element in the context.

Before showing how these suggestions will result in a context schema, the table 4.5 summarizes the categorisation of information of our example.

**Table 4.5.** ***Categorization of information***

| Category | Sets |
|---|---|
| **Very important** | Content (*e-mail*) is not exactly a set but it is the most important information delivered by an e-mail (there are no sets given that would be more-or-as important as the content).<br>Inbox mail (*account*) is so important that it is straightforward to access it first thing in the account. |
| **Important** | Attachments (*e-mail*) are important, but not so much that they would be displayed with the e-mail's content; therefore it can be accessed through an access structure as a part of the e-mail's description.<br>Headers (*e-mail*) are important in some cases, but there is no need to display with the e-mail's content; therefore they can be accessed through an access structure as a part of the e-mail's description.<br>Sent mail (*account*) is sometimes important but not as much as inbox mail; therefore it can be accessed through an access structure as a part of the e-mail description. |
| **Complementary** | Contacts of administrators (*account*) only occasionally one needs this kind of information; it will be accessed by means of an anchor that points to the object that details it. |

Now having in mind this table and the suggestions given above the table, we can form the context schema (figure 4.12). Note: *Anc* means anchored (talking about complementary information); *Idx* means accessible through an accessed structure (talking about important information, or about information that was separated from the original object in the first place).

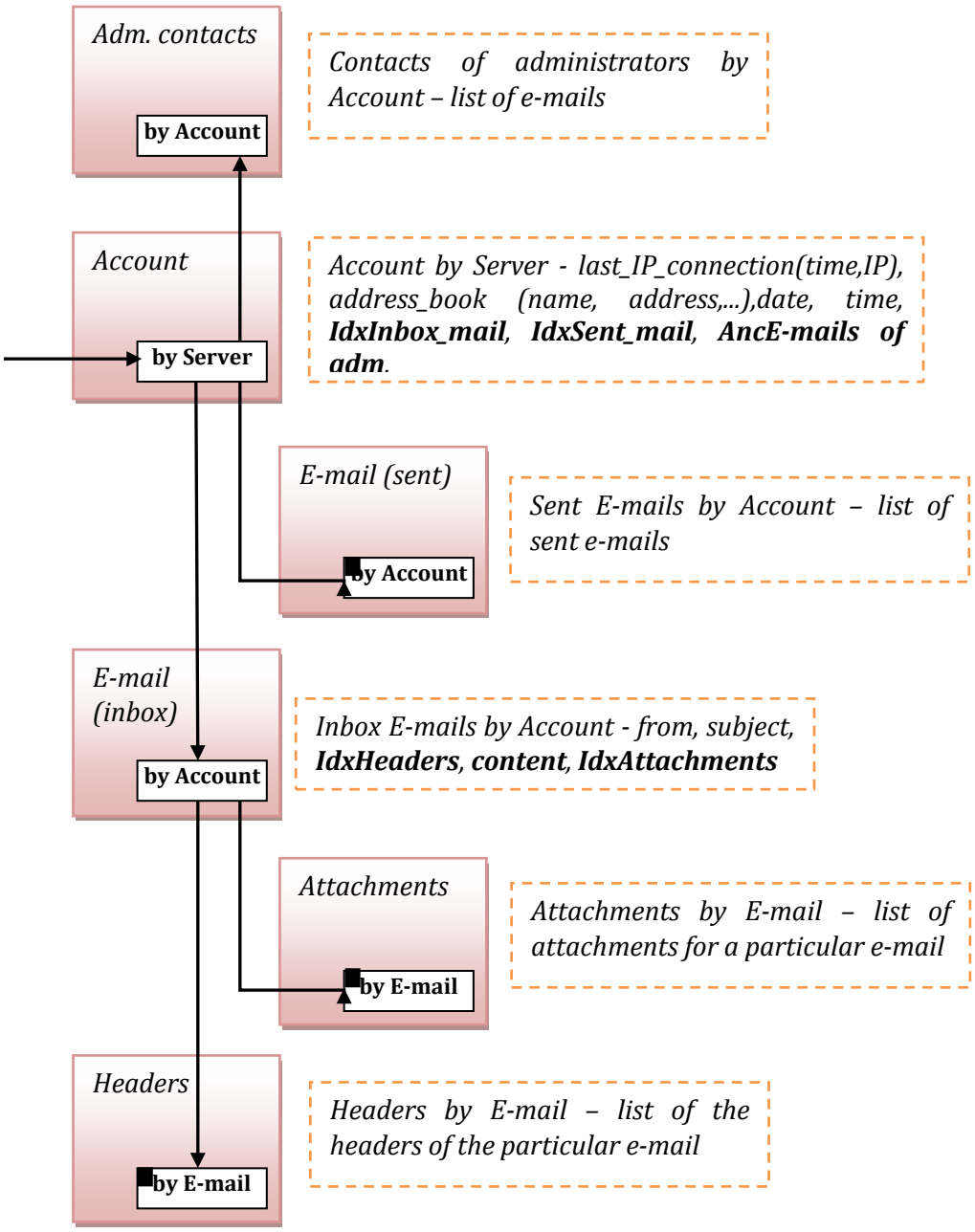



**Figure 4.12.** ***A context schema***

The third rule talks about mapping of a data entry interaction followed by an interaction that presents a set of objects, into an access structure of the set's objects. There are three types of access structures distinguished: dynamic access structure, simple access structure, and hierarchical access structure. The UID we were analyzing so far does not support the dynamic access structures, so for that matter we will provide a new simple UID that supports this aspect (figure 4.13).

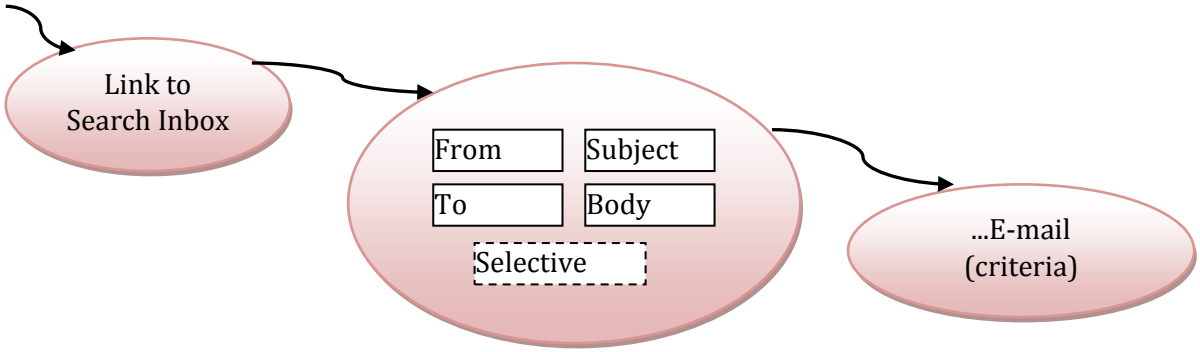



**Figure 4.13.** ***A UID schema***

The UID in the figure lets the user to go to the 'Search Inbox' where one is able to enter whatever keywords in order to get a desired set of e-mails. The overall idea that this given UID represents is that the system is not able to generate the results of user's entry in advance as it is entirely user dependent. For this matter, the given data (keyword) entry interaction and the resulting set (e-mails) are mapped into such an access structure where elements are to be computed from the user's input. In this way we get a dynamic access structure, which in the context schema is indicated by an ordinary access structure element having a heavy side border on the right. An example of such context schema can be seen in the following figure 4.14.

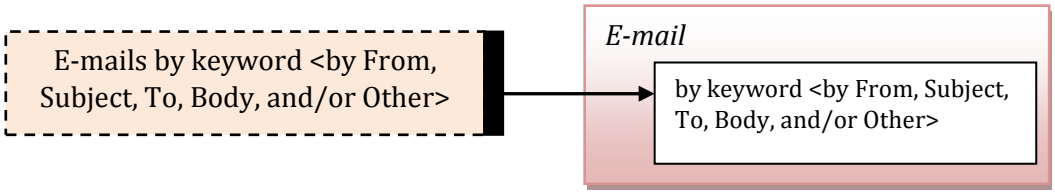



**Figure 4.14.** ***A context schema***

Now if to go back to the UID we were analyzing before this dynamic access structure example (see figure 4.10), in case the system knows all possible valid inputs in advance (which is the case in our main UID), the system then is capable to generate all the options saved in the knowledge base for the user to choose one from (there is a limited choice of Web mail systems the user can select from and the system is aware of all of them). These kinds of input interactions are directly mapped into an access structure. As for the types of access structures, a hierarchical access structure is a sequence of simple structures where the selection of each structure is the input parameter for the succeeding one [1].

In the UID we are analyzing, one has to choose a server first, which is followed by the input of credentials (the input of credentials does not change the context of the object as we are still talking about the server we chose and accounts in that server). For making an example clearer, let's say we chose a Web mail server called 'horde' to go to, and the user of the account is 'renata'. The figure 4.15 below illustrates the results.

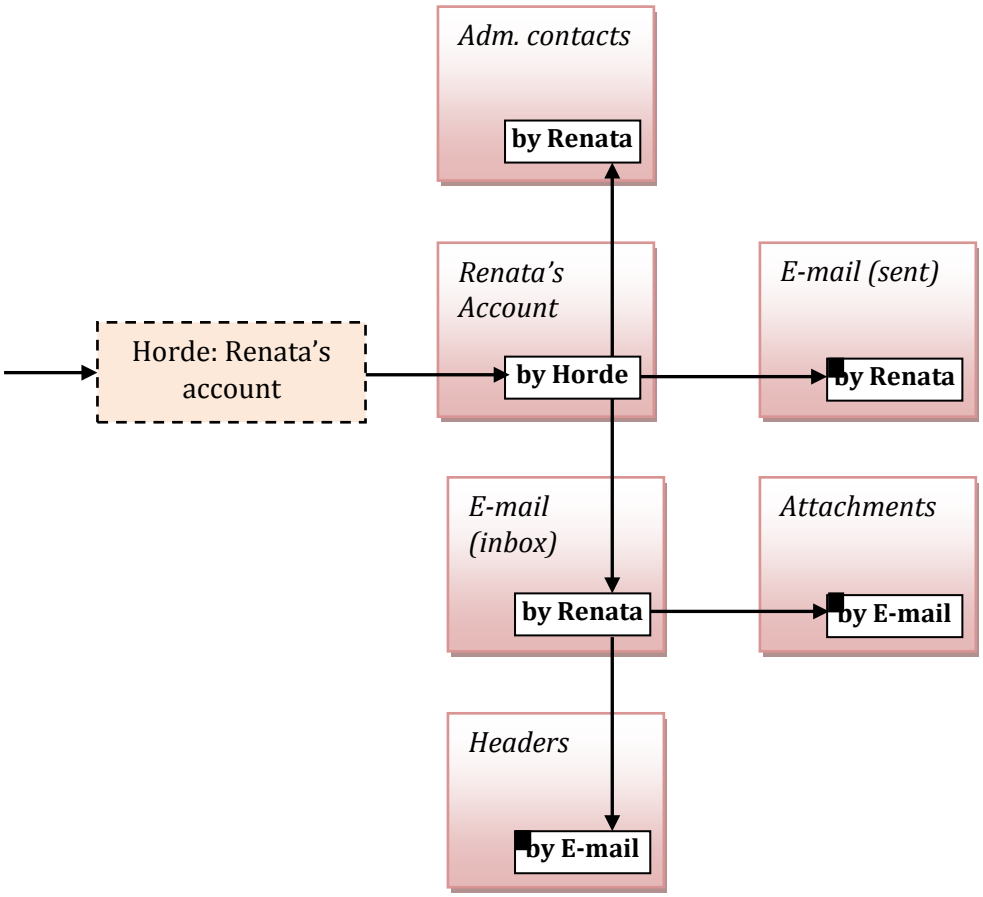



**Figure 4.15.** ***A context schema***

After having done all this work, only one last thing is left to be considered. The thing that we can do exactly with these context diagrams is to write specifications. And specifications have to be written to lists, contexts, and access structures we can find in the UIDs. Specifications are a neat way to put the requirements. This type of documentation is very easy for designers to process and for stakeholders to verify. Overall, context diagrams are for expressing semantic relationships, whereas specifications provide all the necessary information that cannot be expressed in the diagrams themselves.

Anyway, as [1] gives, there are three types of specification cards: for lists, for contexts, and for access structures. Blank examples can be seen in the following tables 4.6-4.8. The names of attributes of the specifications that may be ambiguously understood are explained.

**Table 4.6.** ***A specification card of a list***

| List |
|---|
| **Attributes e Operations[1]** |
| **Conceptual rule[2]** |

**Table 4.7.** ***A specification card of a context***

| Context |
|---|
| **Type[3]** |
| **Parameters[4]** |
| **Elements[5]** |
| **In context Classes[6]** |
| **Ordering[7]** |
| **Internal navigation[8]** |
| **Operations[9]** |

| | |
|---|---|
| **Users**[10] | Permissions[11] |
| **Comments** | |

**Table 4.8.** ***A specification card of an access structure***

| Access structure | |
|---|---|
| **Type** | |
| **Parameters** | |
| **Elements** | |
| **Attributes**[12] | Target[13] |
| **Ordering** | |
| **Users** | Permission |
| **Comment** | |
| **Trace backward**[14] | Trace forward[15] |

1) **Attributes e Operations** – are attributes and operations that are associated with a particular list.
2) **Conceptual rule** defines how to get particular attributes of that list. This is similar to a SQL query 'SELECT ? FROM ? ...'.
3) **Type** talks about type of contexts and access structures. Types can be system dependent but in general, while talking about context one can define them as being *static*/*dynamic*, and talking about access structure – as being *simple*/*hierarchical*.
4) **Parameters** indicate the lowest context level. In the diagrams parameters are given with a word 'by'.
5) **Elements** are contexts or access structures themselves possibly going with some conceptual rule to define them in a particular context (again, similar to SQL query).
6) **In context Classes** indicate other classes that a particular context belongs to.
7) **Ordering** is all about ordering of contexts and access structures. Usually ascending or descending ordering and indication 'by WHAT'.
8) **Internal navigation** is explaining how the data in some particular context can be navigated.
9) **Operations** indicate operations that are associated with a particular context.
10) **Users** define which kind of users can use a particular context or an access structure (a set from the defined actors that go with UIDs).
11) **Permissions** talk about what a particular user is able to do in that context or an access structure. Typically permissions can be read, write, and execute.
12) **Attributes** are attributes of a particular access structure.
13) **Target** is basically indicating the final output of a certain access structure.
14) **Trace backward** – the 'before' step of the access structure.
15) **Trace forward** – the 'next ' step of the access structure.

After analyzing what kind of information list, context, and access structures require, it is time to finalize this part with some actual examples. All the given specifications (figures 4.9-4.11) are based on the same example we have been working so far.

Table 4.9. *A filled specification card of a list*

| List : Adm. e-mails by Account . List_of_emails |
|---|
| **Attributes e Operations : name, e_mail, office_no, contact_time** |
| **Conceptual rule : SELECT a.name, a.e_mail, a.office_no, a.contact_time FROM a: Administrator, e: E-mail WHERE a *owns_ a_certain* e ORDER BY a.name, Ascending** |

Table 4.10. *A filled specification card of a context*

| Context : Account by Server | |
|---|---|
| **Type : static** | |
| **Parameters : s: Server** | |
| **Elements : a: Account WHERE a *is_associated_to* s**<br>***Conceptual rule:* a: Account WHERE (a *has* c) AND (c *is_associated_to* s)**<br>**(by c one has in mind a certain context)** | |
| **In context Classes : –** | |
| **Ordering : by name, ascending** | |
| **Internal navigation : accounts by server** | |
| **Operations : –** | |
| **Users :**<br>**Student, Teacher, Administrator** | **Permissions :**<br>read |
| **Comments :** | |

Table 4.11. *A filled specification card of an access structure*

| Access structure : Attachments by E-mail | |
|---|---|
| **Type : simple** | |
| **Parameters : e:E-mail** | |
| **Elements : a: Attachments WHERE e *has* a** | |
| **Attributes :**<br>**title, size...** | **Target :**<br>download() |
| **Ordering: by attaching order, Ascending** | |
| **Users :**<br>**Student, Teacher, Administrator** | **Permissions :**<br>read |
| **Comment** | |
| **Trace backward: E-mail (inbox) by account** | **Trace forward : –** |

In this part of the thesis we have showed one way of dealing with requirements which is concerned with navigational aspect of a Web system. When requirements given by stakeholders are put into user interaction diagrams, next step to do is constructing context diagrams out of that. The idea is to deal with elements as lists, contexts, and access structures separately. There are some rules that give the guidance. The context diagrams capture semantical aspects of the system. These diagrams, together with complementary specifications, give a good structurized version of requirements. All that has been done at this point has to be given to the stakeholders to verify before designers can proceed with further work.

The navigational approach that will be discussed now is about summarizing navigational requirements into navigation access diagrams. This particular solution talks about ideas suggested by OO-H (object-oriented hypermedia) in [2]. It assumes navigation with respect to the final product.

The key terms in this case are *collections*, *navigation targets*, *navigation classes*, and *navigation links*. The exact meanings are discussed in the table 4.12 below.

**Table 4.12.** ***Key terms of navigation access diagrams***

| Term | Meaning | |
|---|---|---|
| **Collections** | Captures menu concept | |
| **Navigation targets** | Navigation subsystems | |
| **Navigation classes** | Views of conceptual classes | |
| **Navigation links:** | The paths the user has to follow through the system | |
| ***type*** | Navigation links can be of three types: | |
| | **Requirement link** | The beginning of the user navigation through the diagram |
| | **Traversal link** | Navigation paths between information structures |
| | **Service link** | Arbitrarily complex user interface to assign values to the underlying in operation parameters and/or define the visualization of the operation results |
| ***activating agent*** | Can be set to *user* or *system* | |
| ***navigation effect*** | Can be set to *origin* or *destination* | |

This table includes the navigational units that can be found in a navigation access diagram. Before going into an example, it is important to mention that NAD usually does not go just by itself. There is possible a direct mapping between this kind of navigation model and a process model, also given by [2]. The model of NAD for the running example is in the following figure 4.16.

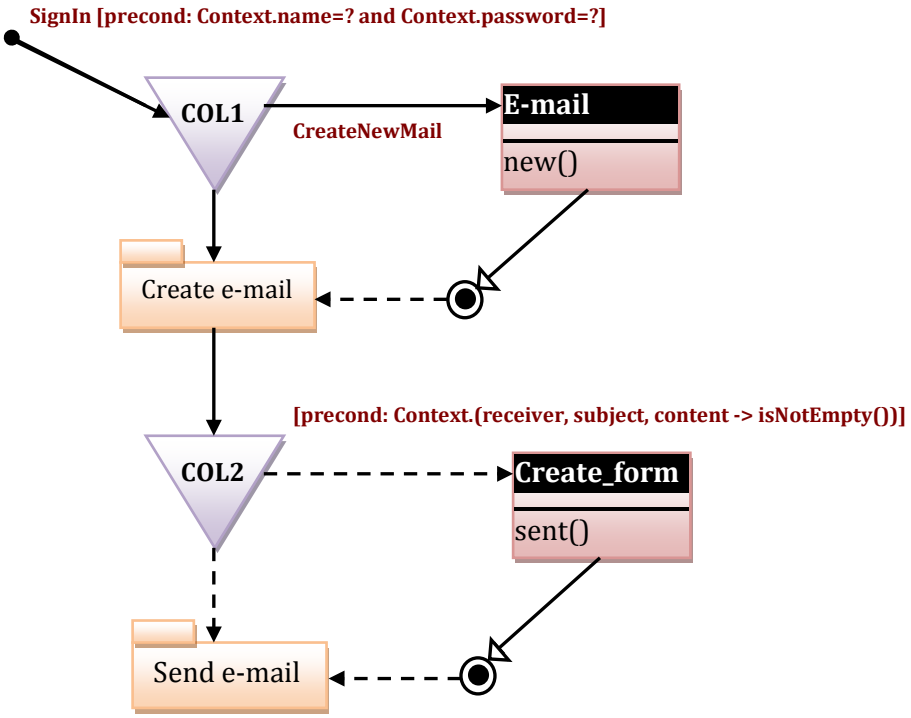



**Figure 4.16.** ***A navigation access diagram***

First, before analyzing what exactly this NAD tries to say to us, let's be sure that we understand exact meanings of the symbols used in the diagram. These are summarized in the following table 4.13.

**Table 4.13.** ***Symbols of navigation access diagrams***

| Symbol | Meaning |
|---|---|
| **An inverted triangle** | An inverted triangle refers to collections (elements of menu) |
| **A class element** | This is a navigational class element that refers to a view of a conceptual class |
| **A package element** | A package element represents a navigation subsystem |
| **A default arrow** | An activating agent set to user |
| **A dashed arrow** | An activating agent set to system |
| **An empty-head** | Navigation effect set to origin |
| **A full-head** | Navigation effect set to destination |
| **A bull's eye** | A transition from activation agent that is set to a user with the navigation effect set to an origin to activation agent set to the system with navigation effect that is set to a destination |

In the example given above, the little round black bullet indicates the very starting point of the navigation. We assume that we start this navigational aspect by signing-in to the mail account; therefore, the required preconditions are correct username and password. As one might notice, preconditions include the aspect of context. This was discussed in the previous navigational proposal.

In the diagram, the inverted triangle 'COL1' indicates a collection, that is, a part of the menu, which allows to choose to create a new e-mail item. The default full-headed arrow to the class element (representing sent mail) shows that the activity was initiated by a user and is set to the destination. This thread of navigation comes back from being user-origin (set) to system-destination (set) where a user (with the help of another navigation subsystem, 'edit e-mail' in this case) is able to do whatever one desires in the e-mail before sending.

After a user is satisfied with all the things done to the e-mail, some preconditions are checked before the e-mail can actually be sent. In this case, the system checks whether a user inserted a receiver, a subject, and the content. Else wise, the system may send a warning message and not send an e-mail before other user actions.

As it can be seen from the example, navigation access diagram shows, basically, navigation within different design items in the system in order to achieve some operational result. In this case – to send a new e-mail. What is more, it captures how everything is related to actual conceptual classes and indicates which actions are done directly by a user, and which are done by the system itself. The best thing in putting requirements in NAD is that stakeholders are able to see interactions with those conceptual classes and to be certain about all the preconditions that the system has to check. The last thing to mention is that these diagrams may be expanded in order to capture all the system's operations, if necessary. We already talked about the difference between user and system operations in part 3.1.

## NAVIGATION ACCESS DIAGRAM, SOLUTION 2

This is a second solution that considers the idea of a navigation access diagram. The solution is also based on the same OO-H approach but this time the interpretation discussed is proposed in [4] and in [17]. Although the overall suggestions are similar, they present some different flavours for this practice.

Even though the key concepts are very similar as in the previous approach, it makes sense to go through them all over again. These are summarized in the following table 4.14.

**Table 4.14.** ***Key terms of navigation access diagrams***

| Term | | Meaning |
|---|---|---|
| **Navigation classes** | Enriched domain classes whose attributes and method visibility have been restricted according to the user access permissions and navigation requirements | |
| ***attributes*** | **V-attributes** | Always visible attributes |
| | **R-attributes** | Reference attributes; these are displayed after an exceptional user's demand |
| | **H-attributes** | Hidden attributes which are only displayed when a specific occasion requires that |
| **Navigational targets** | Group the elements of the model that collaborate in the exposure of a certain user navigational requirement | |
| **Navigation links** | The navigation paths the user can follow through the system; additional information can be added to construct the user navigation model – to | |

| | | |
|---|---|---|
| | restrict or qualify the target information | |
| ***types*** | **I-links** | Internal links – identify the navigation path inside the boundaries of a given navigational target |
| | **T-links** | Traversal links – defined between navigation classes belonging to different navigational targets |
| | **R-links** | Requirement links – the starting navigation point in the entry of each navigational target |
| | **S-links** | Service links – demonstrate the services available to the user type associated to that NAD |
| **Collections** | Usually hierarchical structures (dynamic or static) defined on navigation classes or targets; provide the user with ways of accessing information (concerning behaviour of the link structure and the set of objects) | |
| ***types*** | **C-collections** | Classifying – the object population criteria is predefined |
| | **T-collections** | Transactions – the group of navigation services provided to the user |
| | **S-collections** | Selector – the user should introduce the criteria at execution time |

As it can be seen from the table above, this approach gives somewhat different, but overall similar ideas about the key terms. Nevertheless, things will be clearer after a certain example will be analyzed. Both, [4] and [17] proposes not even the same solution, but the same example. That does not give a better understanding for this matter. Nevertheless, as much as it concerns us here, we will apply the given solution to our running example. The following figure 4.17 illustrates similar idea as the example in the part before (see figure 4.16).

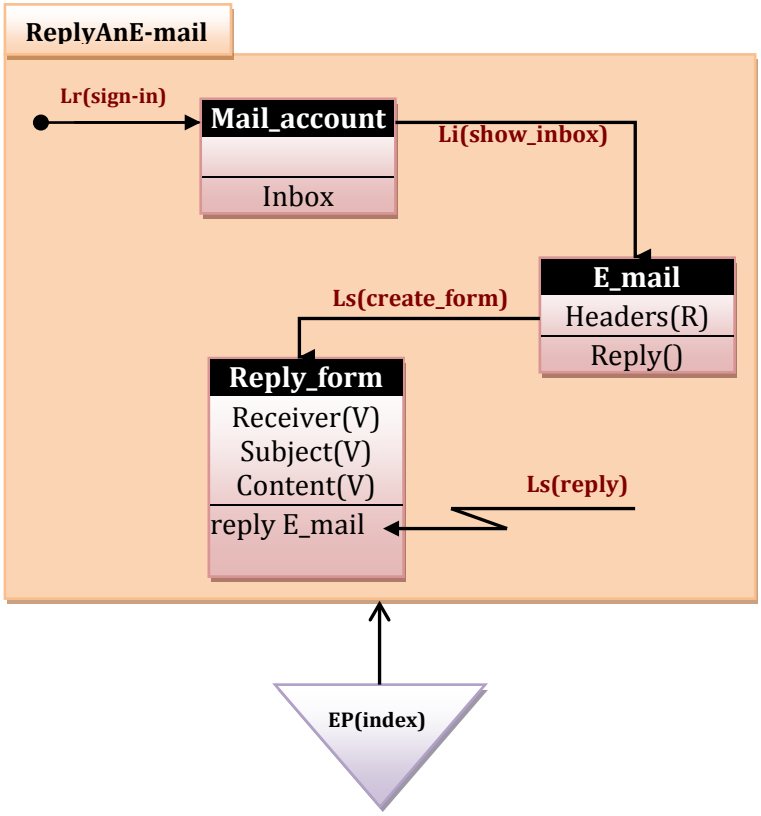



**Figure 4.17.** ***A navigation access diagram***

The table 4.15 below summarizes the meanings of the symbols that this NAD consists of.

**Table 4.15.** ***Symbols of navigation access diagrams***

| Symbol | Meaning |
|---|---|
| **An inverted triangle** | An inverted triangle refers to collections |
| **A class element** | This is a navigational class element that refers to domain classes |
| **A package element** | A package element represents a single navigational requirement |
| **'Lr' full arrow** | Requirement link – a starting point of the navigation |
| **'Li' full arrow** | Internal link – navigation inside the boundaries of a navigational target |
| **'Ls' full arrow** | Service link – services associated with a particular NAD |
| **'Lt' full arrow** | Traversal links – navigation between two classes that belong to different navigational targets |
| **Simple arrow** | A simple communication path |

Having in mind this table containing the summary of meaning of the diagram's symbols, it is time to understand the given example. As it is opposed to the first solution we analyzed before, this one absorbs all the navigation ideas basically in a one single component – the navigation target; that is represented by a package element and is called 'ReplyAnE-mail'. Generally, the navigation target we are analyzing is for navigation within the system in order to reply an e-mail.

An access to the single navigation target is expressed by a collection symbol. At present, it is just a simple entry point to the Web application in general. But, as we know, the connection criterion is predefined, this is called a classifying collection. Now the overall navigation in the navigation target starts where there is an arrow with a bullet-point tail. This arrow shows the 'sign-in action' which is relevant to the domain class 'Mail_account'. What is more, this arrow characterized to be 'Lr' which means that it is a requirement link indicating the starting point of the navigation.

'Mail_account' is associated with another object within this navigation target that is 'E-mail'; therefore it is an internal link relationship. As an example of attributes, 'E_mail' class contains an attribute 'Headers(R)'. This indicates that headers are reference attribute which means that they are displayed only with user's demand. 'E-mail' class has a relation with a 'Reply_form', which is, at this point, just a temporary object. It becomes a permanent 'E_mail (sent)' object only if it is actually sent. But this does not concern us here at this moment. What does in fact concern us is the relation between the 'E_mail' and the 'Reply_form' that is a service link as the relation is explicitly made by the user who is willing to reply to a certain e-mail.

What is more, 'Reply_form' has three attributes as an example of visible ones. These are the most important because in order to send an e-mail there has to be a receiver, a subject, and something in the content field. Furthermore, this 'Reply_form' has a 'reply E-mail' service which is a service link by itself.

The overall suggestion for using this solution would be to make it use-type sensitive. This means making different navigation access diagrams for different roles of users. And, of course, make as many of them as it is necessary to provide a full view of the system. The nice thing about this solution is that navigation access diagrams can be modelled by taking into account separate

chunks of the navigation targets, or putting some together in one diagram if that makes more sense. This shows the biggest difference between the two solutions. That is because in the former solution we had more of a flow of actions done within some parts of a Web application, whereas this method thinks of an application as being one unit with one navigation target. And the question is not which solution for this approach is better; the question is which is superior when dealing with a particular system. This thesis in the end will give some insight for similar questions.

## NAVIGATION MODEL BASED ON UWE

The following method that we will present is proposed in [4]. The idea is originally taken from so called UWE (UML-based Web Engineering) methodology. We are interested at this point in how the requirements can be put into navigation models. The navigation model discussed here is special in a sense that it not only specifies in which objects can be visited while navigating through the application of the system, but also it tries to show how exactly the user can reach desired navigational elements.

For the purpose of modelling the *how* part, this approach includes stereotypes as <<process class>> and <<process link>> additionally to <<navigation class>> and <<navigation link>>. The ideas that these stereotypes encapsulate are summarized in the table 4.16 below.

**Table 4.16.** ***Stereotypes of the model***

| Stereotype | Meaning |
|---|---|
| **<<process class>>** | A class whose instances are used by the user during an execution of a process |
| **<<process link>>** | The association between a navigation class and a process class |
| **<<navigation class>>** | Node, representing a view of a conceptual class in the system |
| **<<navigation link>>** | The association, connecting navigation classes |

This method works by finalizing the navigation model in two steps: the first step is about extending conceptual model with process elements, and the second step extends everything that has been done so far with access elements. As for the conceptual model matter – one can take a conceptual schema. And at this point one can use the conceptual schema that was done in the information modelling part. We will use a similar schema to the one that can be found there but a little bit different in order to show all the relevant modelling capabilities of the method we are now discussing. The schema illustrating the first step of this method can be found in the following figure 4.18.

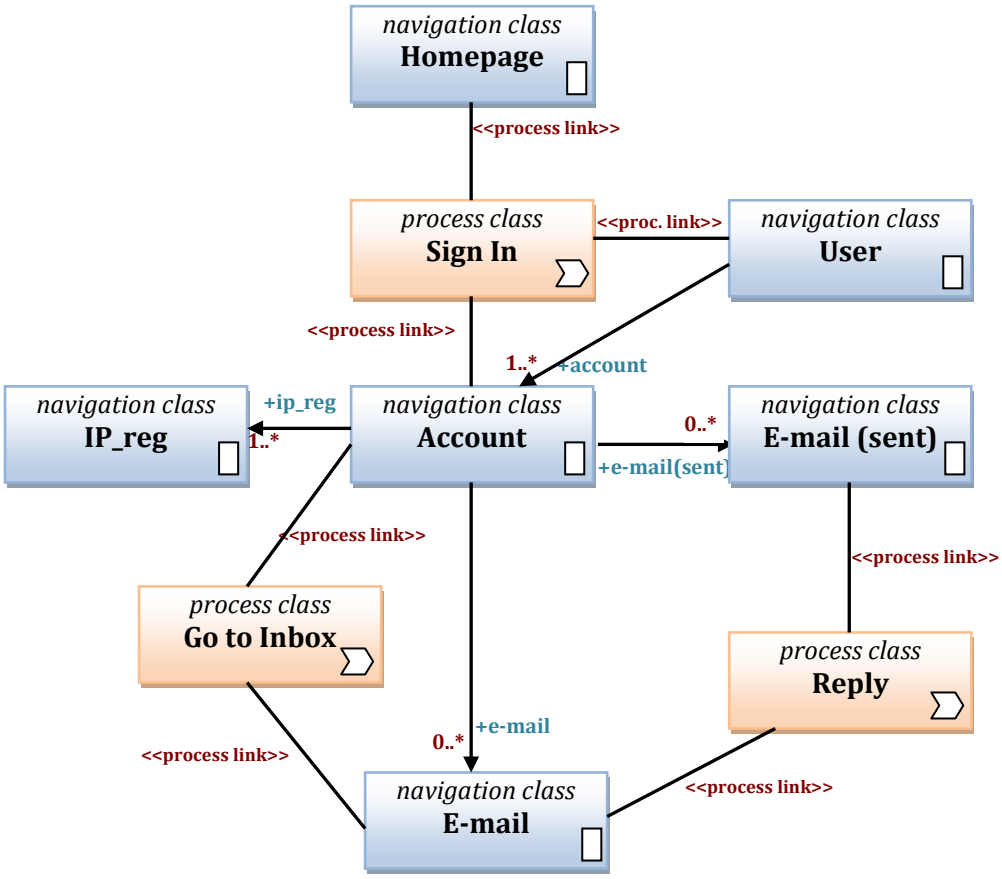



**Figure 4.18.** ***The first step of the method***

To be clear of what is going on in the figure above, the navigation class is supported with a little rectangle in the corner, and the process class is supported with a wide arrow in the corner. What is more, an arrow in between the classes that has no label on it presents a navigation link.

Overall, the given schema can be done directly from a conceptual schema or, as it is known from UML [16], a class diagram. These ideas have been discussed already in the information modelling part having in mind that a conceptual schema in this case basically consists of navigation class and navigation links. If to consider the situation from a general perspective, this might not always be the case but it is likely possible.

Now if to analyze the meaning of this particular schema, we start by stating that we have some sort of Homepage (and a starting point of such a diagram, 'Homepage' for this case, has to have a separately defined navigation class element). While being in a homepage, one can sign in to the personal account. For this matter, process class 'Sign in' is related to the navigation class 'User' and the navigation class 'Account'. As related to the account, IP registry data is linked with this account. With the same idea, e-mails are linked to the account. The difference is that in a system registry data to the user is displayed at once but to see an e-mail one has to go to inbox (which is indicated by 'Go to inbox' process class). When we think of an e-mail, one can make a reply with a help of a process class 'Reply'. And thus we get a connection to the navigation class 'E-mail (sent)'. This connection is only an act of a reply connecting an e-mail that was got from a particular sender.

As it was already mentioned, this sort of a schema is only a first step of the intended navigation model. The next step would be to expand this very schema. This is suggested to do by adding

three kinds of stereotyped classes: <<index>>, <<query>>, and <<guided tour>>. It's not that these must be all in the schema but as we are talking about Web based information systems, they usually are; especially stereotype class <<index>>. We will use this particular stereotype to expand the navigation model presented in 4.18. The result can be seen in the following figure 4.19. Just to be thorough, these three stereotyped classes together are called access structures.

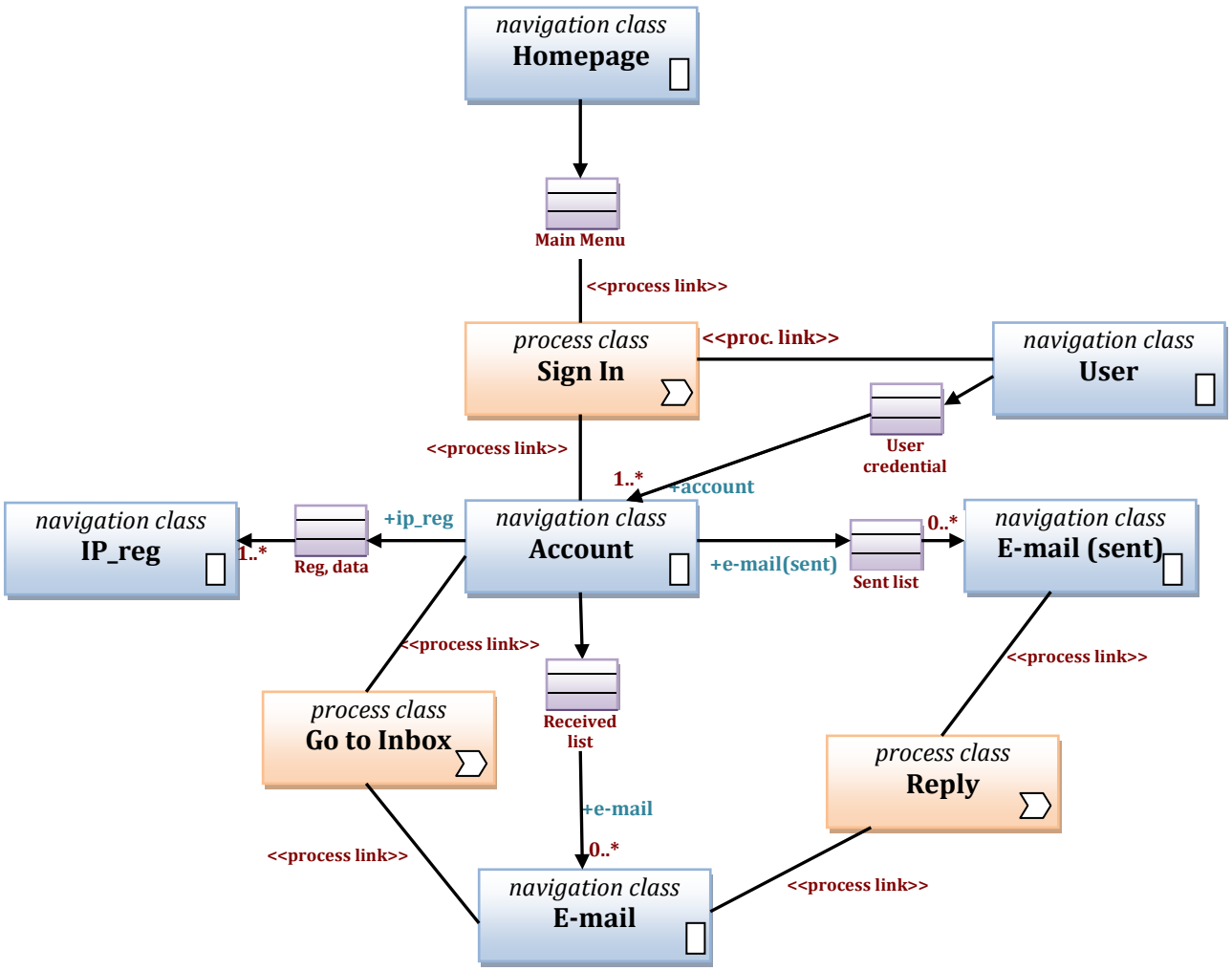



**Figure 4.19.** ***The second step of the method***

As it can be seen in the given diagram, it is similar as before just access structures added to it. All the structures for this matter are of type <<index>>. The overall idea is to basically explain in which way a navigation class access another navigation class. For example, a navigation class 'Account' access a navigation class 'IP_reg' by asking Registry data. What is more, if one would like to access some data from navigation class by a search or filter activities, these would require an access structure of type <<query>>.

Basically an idea of this whole method is to embrace the proposal of transforming a conceptual schema into a navigation class diagram supported with process classes and access structures. This way allows one to actually see not only associations between the navigation classes but also to see what actions have to be done in order to access them and, what is more, the way of how the navigation classes access each other.

This navigational model tries to capture hypermedia requirements. It is presented in [23]. Although the overall suggestion that is proposed focuses more on service modelling, but at this point to be thorough, it makes sense to discuss those aspects that are directly important when dealing with navigation modelling in general.

So the idea is that for every kind of a user there should be made a navigational map, which would capture the structure of a Web based system. The map would look like a directed graph where nodes represent navigational context (that refers to a Web page) and edges are navigational links (referring to links that connect particular Web pages). In the beginning of this part of the thesis we have already discussed a lot about the context in general. This particular navigational model aims to make a distinction in the context not so much by data items as we had before but by particular Web pages. That is, an assumption is made in advance that different pages will refer to different contexts.

One more thing that is suggested in [23] is idea about operations of services. These will be discussed in later parts of this thesis. But for now one could remember that navigation can be used not only to navigate between Web pages or data structures, there is always a possibility for users to invoke the operations of services when navigating in a system.

What is more to the idea of the context, for this matter, is that the context can be built of a set of interaction units of information and functionality. These both interaction units are simple units that could be added to composite units with an aim to support an aggregation, a typical requirement of complex Web sites. This is the proposal by [23] and it is captured in the following figure 4.20.

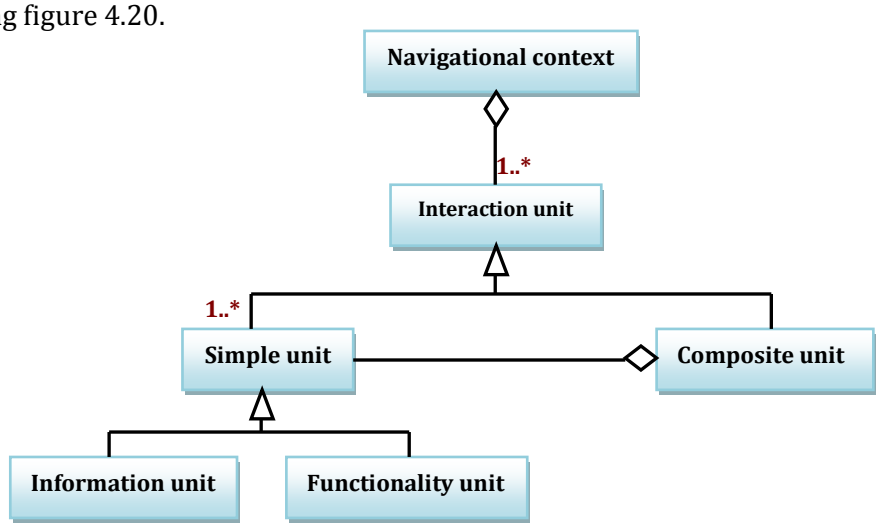



**Figure 4.20.** ***The summary of the proposal***

Now it is important to understand information and functionality units a little bit better. These are considered as the key components that form a context of a certain Web page.

*Information unit* is such an interaction unit that can be defined as a view of attributes and operations of a class in the class diagram. Again, we have to go back a little into information modelling part, as this refers to conceptual schemas that we already discussed. Having in mind

the similar approach, the ideas behind this particular method suggests something new. The information unit can consist of a main information unit and some complementary. The table 4.17 explains the meaning of these two units.

**Table 4.17.** ***The information unit***

| **Information unit consists of:** | |
|---|---|
| **Manager information unit** | The principal information of the context |
| **Complementary information unit** | There can be a set of complementary information units that provide additional information about the context (it is possible for this set to be empty) |

Manager information unit and complementary information unit always have a relationship that can be of two kinds. Either it is a *contextual dependency* relationship, or a *context* relationship. The further explanation can be found in the following table 4.18.

**Table 4.18.** ***The relationship***

| **Relationship** | | |
|---|---|---|
| **Contextual dependency** | Represents requirements of recovery of related information between information units | |
| **Context** | In addition to recovering information, this relationship offers navigation *capability* to target context | |
| | **Capability** | |
| | **Context attribute** | Indicates the target context |
| | **Link attribute** | Indicates the attribute link to the target information unit |

An example summarizing the ideas above is in the following figure 4.21. We stick still on the running example. In this case we choose to analyse a Web mail account class and an e-mail class that is related to this account.

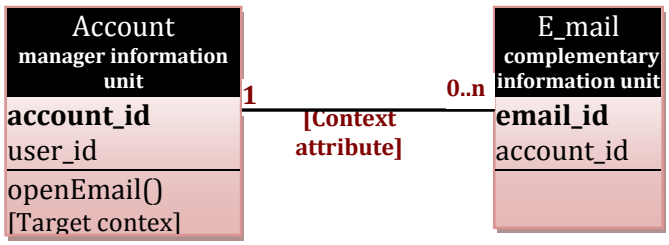



**Figure 4.21.** ***An example of the proposal***

In the example above we have one manager information unit (Account) and one complementary information unit (E-mail).These are related with a context attribute relationship. With respect to the account, when performing an operation openEmail() with indication that it is a target context, the idea is that in this way we get a new context. The new context is an e-mail in the particular account.

*Functional unit*, for this matter, is an interaction unit that captures own or external functionality when the user is navigating. Each functional unit has a view of some operation of the services that are proposed by the system. Anyway, what is important at this point is that when an

operation of a service is executed, navigation to the target context could be possible. The suggestion that [23] proposes is: if that execution returns a response, the context remains the same; if there is no response – the navigation is transferred to the target context. An example of a functional unit can be seen in the following figure 4.22.

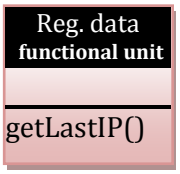



**Figure 4.22.** ***An example of a functional unit***

Relationships between an information unit and a functional unit capture requirements of information visualisation and the following execution of a service operation. What is more, with regards to information and functional units, there can be two cases distinguished: attributes of the information unit can be values of formal parameters of functional unit operations; or an information unit has an operation with a target context including a functional unit. An example of a full schema of navigational context can be found in the following figure 4.23.

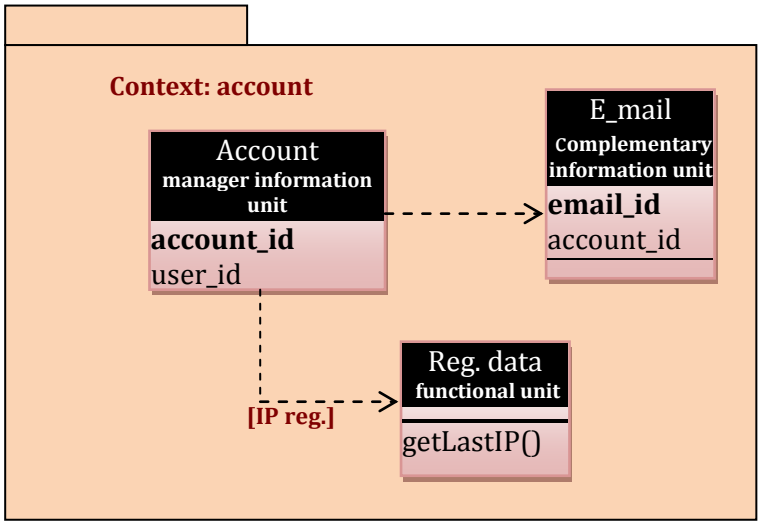



**Figure 4.23.** ***An example of the proposal***

The given example shows that we are modelling an account context, where 'Account' information unit is related with 'E-mail' information unit and, what is more, there is a relationship with a functional unit that provides registry information about an IP address.

The overall idea of this method is that one can model context for every important information unit that, in a certain context, is taken as a manager one. Many of these context diagrams show how the navigation, with respect to contexts, in the whole system is done. This is an alternative way of expressing the navigation to what we have discussed before.

In this part of the thesis there were five proposals of requirements processing into navigation models suggested. The outcomes of these methods are different, therefore, a person responsible for the processing, should choose one that fits the most for the system under development according to the characteristics these methods provide.

The UID based approach and the universal navigational model (presented last) both take into consideration the context of navigation. The former method allows building access structures directly from UIDs. What is more, it is a very thorough method allowing to write detailed specifications for lists found in the system, contexts, and access structures. While this comprehensiveness could be very appreciative both by designers and the stakeholders, this level of detail providing may not be necessary if the context aspect is not so important or if stakeholders are not interested with that kind of scrutiny. Now the latter method suggests associating different contexts of the system with different navigational Web pages, assuming that each page is related with a different data item. The overall idea is to make navigational context diagrams for different classes from conceptual schemas. This way of working allows keeping a thread of context from different perspectives and seeing how the navigation is done.

The two solutions talking about building navigation access diagrams work on similar concepts but provide different interpretations. The first solution is all about showing how different navigation subsystems interact with classes from a conceptual schema, and the sequence of that interaction is shown as well. The second solution concentrates more on applying relationships of classes and the other relevant stuff within one navigation subsystem. Many on these would form a navigation system of the whole system.

The last approach to discuss is a navigation model based on UWE. The suggested idea is all about representing a conceptual schema together with classes representing processes that are directly working with information units. What is more, it allows showing directly in which way those information units can be accessed.

## 4.3. OPERATION MODELLING

This part of the thesis is all about capturing a sense of functionality. There are two main methods proposed of how to work with requirements in order to get the desired result. They are not so much similar as fulfilling each other. Generally, operations are much related to services but these will be discussed in some other part of this thesis. This particular part talks about all relevant ideas of operations that touch sense of functionality in general.

The first approach that is proposed by Schewe, K., Zhao, J., and Thalheim, B. in *Quality Assurance in Web Information Systems Development*[7] introduces an idea of building so called task-goal refinement trees that give an overall understanding of the system's functionality along with expectations towards different parts of it. The very important thing of this method is that after

the trees are built, stakeholders can see very clearly who can do what exactly in the system and what the system itself suppose to do.

The second approach, that [3] presents, gives a deeper look of what is going in the system. The idea is to catch the dynamics of operations using a sequence diagram method from UML. What is the most important at this point – that this technique shows how messages are exchanged with the user, and what is more, the exact sequence of them.

## THE GOAL-DRIVEN APPROACH

The idea of the goal-driven technique, as its name indicates, is based on setting goals, namely, by defining a mission statement. Mission statement should in general terms describe what a Web information system is all about, describe expected content, functionality and context. What is more, it should include description of actors (or roles as we discussed a little in 4.1), tasks and goals. In addition, some sketching of ambiance and desired atmosphere would also be considered necessary.

In general, while talking about goal-driven technique, we have in mind two main concepts: a provider (describing *what* kind of content the system offers), and a user (describing *actions* which indicate the functionality of the system). The scheme for all the parts discussed above that was adopted by [7] is of form $P^{W}2U^{A}$ and is called a *brand*. It can be seen in figure 4.24, and it can be read as: 'from' the provider 'to' the user. Superscript letters indicate extensive information respectively about the provider and the user.

$$P(provider)^{W(what)} \rightarrow U(user)^{A(actions)}$$

**Figure 4.24.** ***The brand:$P^{W}2U^{A}$***

*P* stands for 'provider' and describes the system itself together with the expected content. *W* stands for 'what' and adds more details about what kind of content the system will provide. And by content one has in mind all the information that the system can give. *U* stands for 'user' and explains to whom services provided by the system will be directed. In this case 'to whom' can be understood as for which kind of roles as in [1], and services that the system provide can be understood as operations, as it was in [3], where we distinguished that there are two kinds of them: user operations and system operations. Now *A* stands for 'actions' and ads more details about all the functionality of the system. All this structure, the brand, is complemented by the mission statement.

The overall idea of the *W (what kind of content provided)* part is to become a set of nouns defining content and for the *A (actions that represent functionality)* part to become a set of verbs defining functionality. These keywords can be refined and put together in order to make semantic relationships. These relationships can capture specializations, or associations mapping global context with details. Refined nouns especially should indicate navigational features and somewhat structure the content. What is more, it is important to relate functionality with

content: map type of contexts with particular contents; map contents with particular activities and the other way around; determine the order in which the content can be used by an activity.

[11] also is up for hierarchical structures. The beneficial thing about hierarchies is that they enable a possibility of using different levels of granularity while presenting information, thus allowing to switch among these levels if it is needed and make modelling more understandable.

The suggestion that [7] has in achieving all that was stated before is by making refinement trees. That is refinement trees for nouns and verbs taken from the brand, where they start as being root nodes for those trees. These trees together are called an *utilisation space*.

At this point we will stop for a moment and figure out what the brand is all about. For that, we will use the same example of *www.nfit.au.dk* Web system. To make things clear at this point, we will take only some particular features of that. As it was mentioned before, we have to describe the content and actions related with this system. Figure 4.25 sums up chosen aspects.

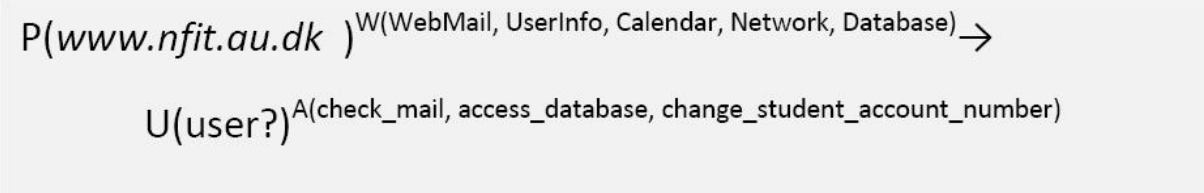



**Figure 4.25.** ***An example of the brand***

And so we have an example of the brand. It is worth mentioning one more time, that it represents only a little part of what the system can provide and do. Describing the whole system in this way would be out of scope of this thesis and would only complicate the illustration of the main relevant points that have to be demonstrated.

The left-hand-side of this expression talks in simple nouns of the basic provided content, and the right-hand-side gives examples of some actions that users might like to perform using this system. There is a question mark in the 'user' entry because the idea of the brand as well as everything that was mentioned helps to identify roles of the users. By following the instructions of [1] we already did some work of that in 4.1, therefore, similar aspects of this technique have to match. Nevertheless, mail checking action can be performed by a student, a teacher, and an administrator; accessing database is allowed to teachers and administrators; and to change student's account number can only an administrator. Having in mind all this we come to a concept of *utilisation portfolio*, which concentrates on the *U (user's)* part of the brand. It is all about classifying users as actors according to which role they perform, their goals while using the system, and their tasks that have to be executed in order to achieve their goals. Tasks, in this case, correspond to actions in the brand and they were already discussed.

So it wouldn't be confusing, it is worth mentioning one more time: we classify users as actors and an actor is described by a triple {role, goal, and task}. Tasks are actions in the brand, roles are attached to the concrete tasks and from the above example we have roles: {student, teacher, and administrator}, goals are attached to the roles and we are missing them at this point but we will come to that very soon. What is more, [11] argues that actor modelling can lead not only to roles, profiles, and goals, but also to preferences,

obligations, and rights. This information is also important and can be used in later stages of modelling.

While making the refinement, tasks are refined into sub-tasks to the level of an elementary task which can be associated with a single role. Refinement of goals is also made, therefore, to make a consensus, sub-tasks have to refer to sub-goals. The idea is to make a *task-goal* refinement tree (graph) to represent all the dependencies. There are three types of nodes distinguished: for tasks, goals, and actors. This gives different kinds of edges: task-goal, goal-goal relationships, involvement of an actor in the task, and, what is more, we have sub-tasks and task specializations.

Having in mind the above given brand, we will try to give an example of a task-goal tree. In fact, for this matter, two of these trees will have to be built. One tree will be concerning Web mail in the system, and another one concerning the database given by the system. Although it is suggested to use actions from the brand as root nodes for task refinement, it is not always the case that we collect actions serving as roots; many times we collect sub-tasks that can be merged into a tree with some other root task. In case one wants to be very thorough, one may start with goal 'using the system' and the following root task 'open the system' but this approach will not serve its exact purpose. Overall, everything depends on how big trees one wants to build. Therefore, as long as the system's properties are visualized correctly, the rules of how to exactly use this technique are not so strict. Task-goal trees are given in figure 4.26 and figure 4.27.

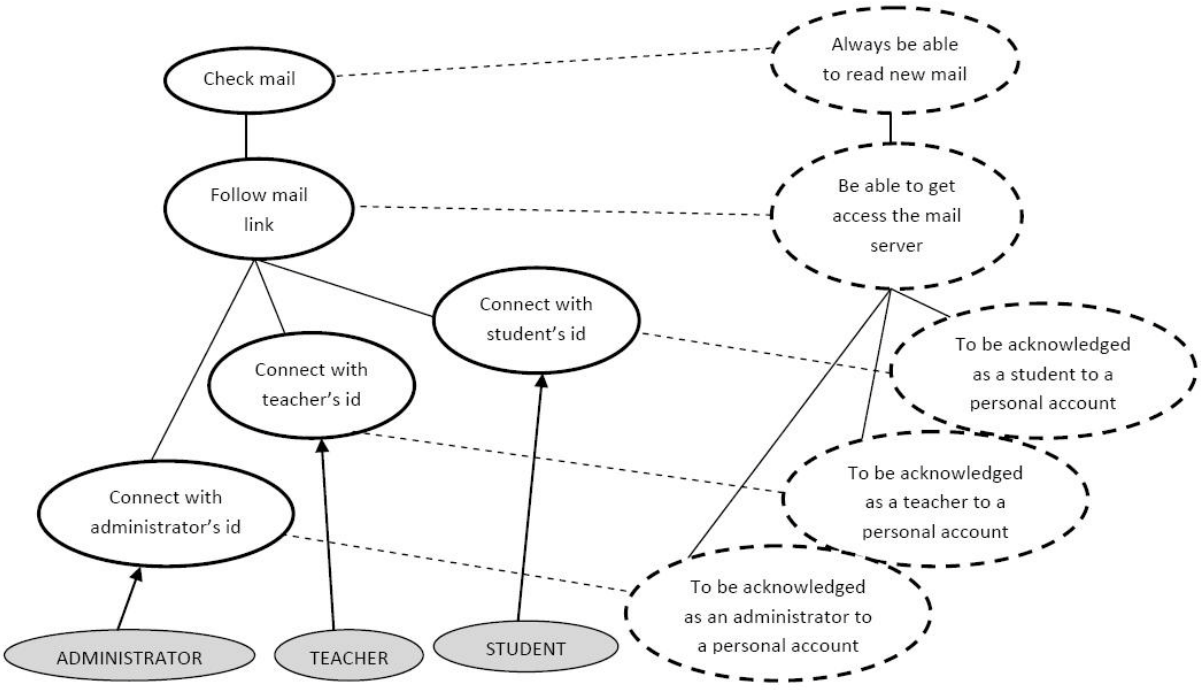



**Figure 4.26.** ***Task-goal refinement tree concerning Web mail system***

The way the tree is represented in this example is just for better understanding purposes. Nevertheless, there are no strict rules of how to exactly do that. In this case, tasks are represented with full-line circles, goals with dashed circles, and actors with grey-filled circles. What is more, nodes and sub-nodes are connected with straight lines, goal-task edges are dashed lines, and the role of an actor is applied with an arrow to the elementary task. As it was

mentioned already, elementary tasks have to be associated with a single role. In this case, even if the same task in fact can be performed by more than one role, they have to be separated for clarity purposes.

In the refinement tree that is given above we have a main root task – to check the mail, which further consists of: following the relevant link and connecting to the account with relevant credentials. And as we have already established, mail accounts can be used by students, teachers, and administrators. What is more, every task has an associated goal with it, which mainly arises from corresponding roles. Goals and tasks sometimes can appear very similar. The goal generally is some end-point in a space that one is trying to achieve, whereas a task is an action that takes one to that particular end-point. In the example our main goal is basically to read new e-mails, whereas the action that lets us to read the new e-mails is checking the mail box. Normally, when using a system, a user has one overall goal and does not think about sub-goals that go with it. This task-goal tree forces on actually considering them. For example, if a mail server is crashed, and the user cannot connect to it, the goal in the user's mind is to actually get connected to that server. But that is not somewhat a new goal; it was all along there, although very much implicit in the root goal. This way of matching tasks with goals can better show users' expectations and can help the designer to better notice where assistance to the user or alternative solutions are needed in case some particular goal at some point cannot be achieved.

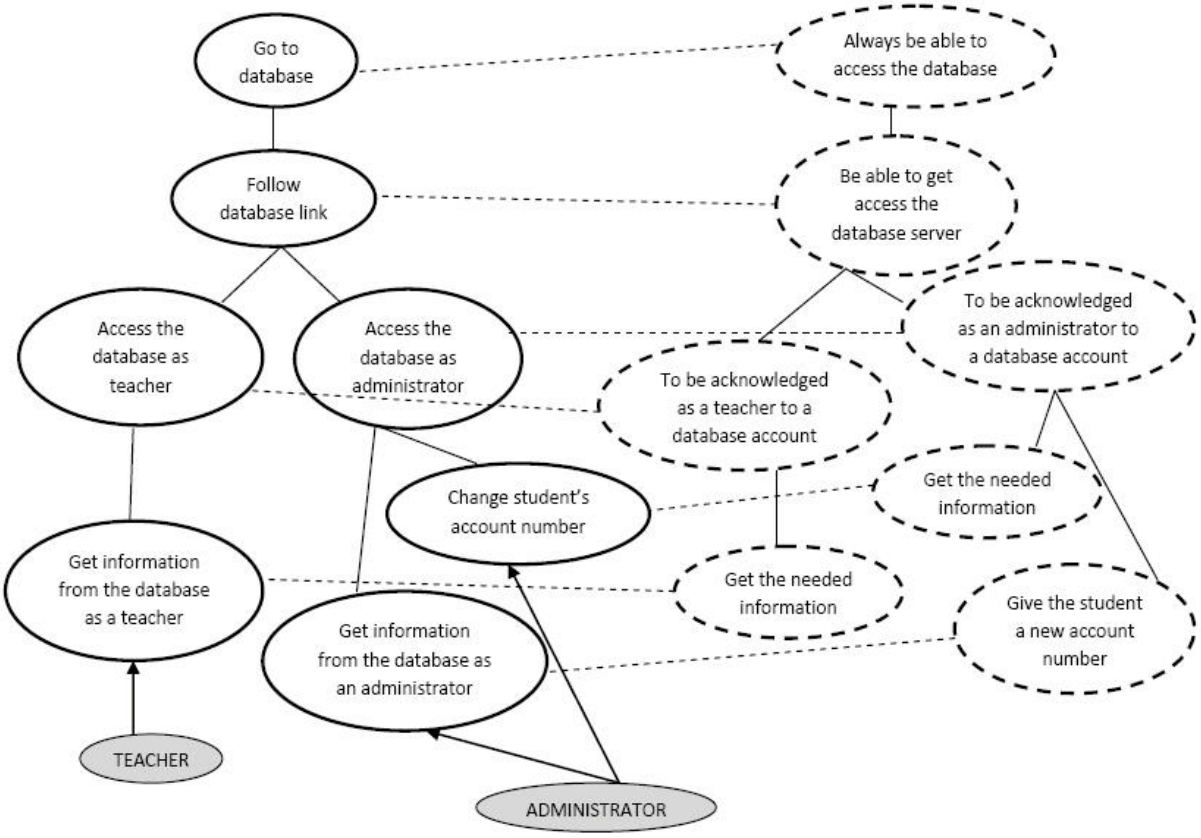



**Figure 4.27.** ***Task-goal refinement tree concerning database the system provides***

This schema in figure 4.27 captures two actions defined in the brand: accessing the database in order to get information and accessing the database in order to change student's account number. The basic thing that one might notice that it is a quite abstract schema because in

everyday systems there are many more steps or phases of the system to deal with. Nevertheless, one can choose the level of abstractivity according to ones needs. There are no rules, only an advice – to keep things simple. The most detailed refinement tree is not necessarily the one that provides the most relevant quantity of information. Sometimes complexity can only overburden other stakeholders for which the schema is meant to and require a lot of time to analyze it.

## MODELLING OPERATION DYNAMICS

There is a couple of interesting thoughts that [3] gives while explaining why Web operations are different from ordinary operations of any other kind of a system. These are those two ideas:

- 'Web operations do not operate on generic data structures, but on peculiar structures that capture hypermedia specific abstractions.'
- 'Most operations for web applications are something different from operations in conventional software: they can be invoked during navigation (and may need to take into account the navigation context as an implicit parameter), and they may require some navigation steps to be completed.'

Now, these ideas are not significantly new, we have already talked about similar things before. Nevertheless, one more time it is important to mention that all different types of requirements modelling techniques are very tightly connected. That is talking about navigation, we cannot escape talking about information; talking about operations, we cannot escape talking about things like navigation or services; and so forth.

There are two ways of invoking operations that [3] considers. One of them is when an operation invoked from an object of a given type, the other one is when an operation is invoked because of navigation in the system.

*Invoked from an object*. If to talk about operations that are invoked from an object of a given type, these types in Web based systems are entities, collections, and associations. The invocation has to be done regardless of how the user has reached that particular object during a session. An idea that supports this kind of object invocation is so called *free navigation*. Free navigation is all about giving the final destination of some navigation steps without indication what the specific path is. This idea helps to avoid unnecessary navigational aspects and it can be considered as being a shortcut which makes it easier to identify some fixed points that have to be reached by a user in order to complete an execution of a certain operation.

*Invoked from the navigation*. An operation can be invoked in a certain navigational context, or it can be required for a user to reach a given object under some constrains. The second idea is considered to be a *constrained navigation*. This allows specifying some specific paths as a precondition for invoking a certain operation, or for completing an execution.

Now, as [3] suggests, all the requirements concerning Web operations can be captured by an extended idea of UML interaction diagrams.

Before getting into a real example of this, it makes sense to talk a little bit of what in general an interaction diagram (sequence diagram if to be precise) can offer to us. There are some ideas borrowed from [16].

**Table 4.19.** ***Symbols of a sequence diagram***

| Symbols | Purpose |
|---|---|
| **A vertical dashed line** | This is so called lifeline. It represents an existence of a certain object over a period of time. |
| **Aligned rectangles** | Aligned rectangles represent objects that exist for duration of an interaction. |
| **'create' message** | Indicates an object which is created during an interaction; a rectangle that represents a new object is not aligned with the others. |
| **'destroy' message** | Indicates destruction of an object; the lifeline ends with this message. |
| **Underlined object name** | The fact that an object in a rectangle is underlined means that the interaction represents the history of a specific object. |
| **State change icon** | If an object changes state, or values of attributes, or roles, a state change icon can be put on its lifeline where it starts. |
| **Tall thin rectangle on a lifeline** | This is so called a focus of control; the period of time during which a certain object performs an action. |
| **The top of the thin rectangle** | Aligned with a start of an action. |
| **The bottom of the thin rectangle** | Aligned with an end of an action. |
| **Stick arrowhead line** | Indicates an asynchronous message. |
| **Filled triangular arrowhead** | Indicates that a message is synchronous (a call). |
| **A dashed arrow with stick arrowhead** | A reply to a synchronous message. |

The table 4.19 above summarizes all the things that are relevant for an ordinary sequence diagram taken from an UML approach. Now let's go back to what [3] suggests. For the explanation would be understandable, we will analyze an example about the Web system we were using so far. All the things that are different from the ordinary approach will be discussed later. The example is illustrated in the figure 4.28 below.

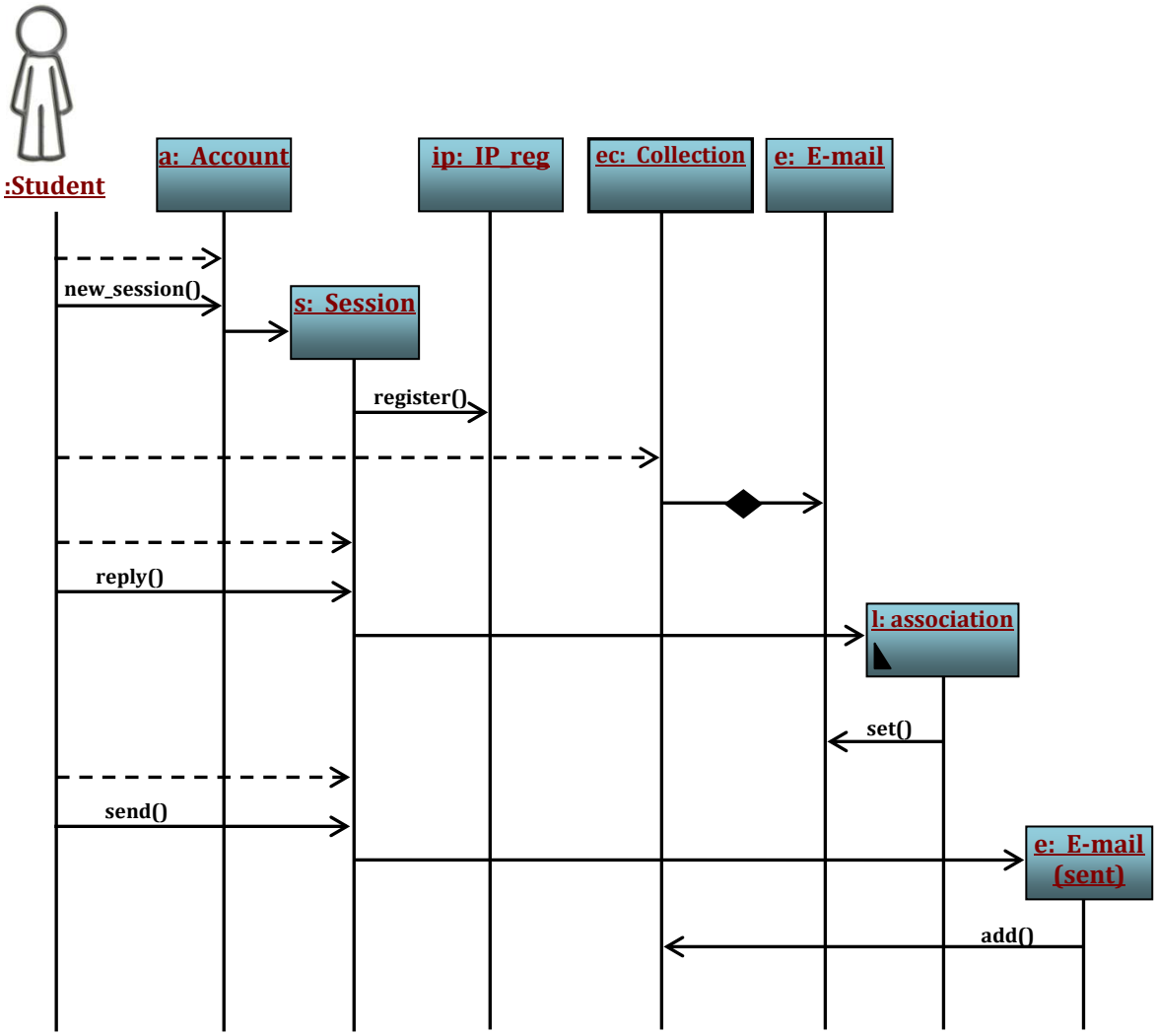



**Figure 4.28.** ***An example of a sequence diagram***

The table 4.20 below summarizes some symbols that are relevant for our case now but is not a part of an actual UML approach.

**Table 4.20.** ***Relevant symbols***

| Symbol | Purpose |
|---|---|
| **A dashed arrow** | It shows that there is free navigation |
| **A normal arrow** | A message within a system |
| **An arrow with a diamond** | This indicates constrained navigation |
| **A thick-border object box** | Represents a collection of objects (or lists as we had before) |
| **A full triangle in an object box** | Indicates that an association has been made |

The figure above (that illustrates an example of a sequence diagram applied to operation modelling) starts by indicating that the sequence is initiated by a student. This in the schema is given as an object but, having in mind that it is a role that we try to express, it is put not in a rectangle but as a symbol of an actor (a student in our example). So the student creates a new session by freely logging in to ones account. This action is registered in the IP registry. After that, one can freely access a collection of e-mails and then navigate with constraints to a particular piece of e-mail. The constraints for this matter are some sort of a criterion according to which that particular e-mail was selected. This kind of a sequence diagram is special in a way that it does not require to show how long a certain object is performing an action; what is more, the idea is that a user can work with Web system's elements by using a session element. For this

matter, a session can be understood as a period of time during which a system is actually used or one is allowed to use the system. Usually, when a session ends, one has to log-in to the system again. That is, of course, if a notion of sessions is in a particular Web system. Nevertheless, if one has in a system more things to do than just to read given information, sessions are usually used. So, after choosing a particular e-mail, student can reply to it. The system then would create a reply form and associate it with that particular e-mail. After the user is satisfied with a reply form, one can actually send it and the system would add this reply e-mail to the collection of all the e-mails.

This way of modelling operations is using an idea of a sequence diagram which helps to see the way the system works from timed sequence point of view. Sometimes it is a very useful technique to put requirements in this way. Nevertheless, this does not show exact relationships among the objects themselves, therefore, it cannot go alone as method for modelling all the system's requirements.

### DISCUSSION

The approaches of this part present functionality aspects of the system. The task-goal refinement trees model the requirements of the system in a way that all the tasks would be covered together with information of the roles that can perform those tasks. It is worth mentioning, that the goal-driven approach is a very thorough one. More than just capturing the functionality in this way, it models all the expectations towards the system by defining goals not just for the system but for all functional aspects of it as well.

The operation modelling approach gets into the way an operation itself is performed in the system and how it communicates with all the parts of the system during its performance. The thing that is very useful with this – not only messages among the system's parts are modelled, but the sequence of those messages as well. This technique contributes very much to the goal-driven approach in a sense of granularity. Together these methods cover the functionality of the whole system very well.

## 4.4. BEHAVIOUR MODELLING

Behaviour modelling is all about showing how the system behaves in general, and how it reacts to users' inputs. These requirements are usually put into process models. This part of the thesis takes into account two process modelling solutions that are both presented in [2]. The first given solution is all about expressing system's process models through activity diagrams. These are similar to the activity diagrams that UML suggests, but with some adjustments. Now, the second solution is based on UWE method that we already heard about before. This time behaviour is captured in activity diagrams but in different version of them. In this case, activity diagrams are enriched with entities representing some behavioural aspects of the system.

The following approach presents behaviour modelling. Behaviour modelling is captured in process modelling (or also can be called task modelling) as it is given in [2]. The overall idea is to use the proposal of UML activity diagrams. The thing with activity diagrams is that there are actions modelled and the transitions are triggered by completion of those actions. In this way requirements can be put in many different alternative system's process diagrams in order to reveal the full behaviour of the system.

Different process diagrams can be based on different use cases, different packages, different operations, or there can be sets of those depending of what exactly a designer wants it to represent. Before we go into details about a specific proposal given by [2], let's peak to some overall ideas that actual UML has to suggest about activity diagrams. These are taken from [16](table 4.21).

**Table 4.21.** ***Symbols of an activity diagram from UML***

| Symbol | Meaning |
|---|---|
| **A little black circle** | Captures the initiation of a certain activity |
| **A rounded-corner rectangle** | A so called activity node; it is an organizational unit within an activity: these are actions or groups of other activities |
| **An arrow** | In the activity diagram it is called a flow arrow; this passes the control from one action to the other |
| **A little black circle inside a bigger white one (bulls eye)** | Captures the completion of a certain activity |
| **A diamond** | A diamond represents branching of a flow: it can be either a Boolean expression or 'else' branching; what is more, a branch symbol can be used to merge back flows together |
| **A thick horizontal or vertical line** | This is a so called synchronization bar; it models flows that are concurrent to one another; it can split flows (forking) or join them (joining) |
| **Partitions of activity states** | These are swimlanes: grouped activities that share some organizational properties |
| **Rectangle representing an object** | In activity diagrams objects that are related with certain actions can be involved |

The proposal that we are discussing right now is not so much different from the original UML approach. It is based on modelling processes of some certain use cases, these we discussed previously in information modelling part of the thesis. So it would be clear what it is all about, the following example in figure 4.29 illustrates an activity diagram of our running example.

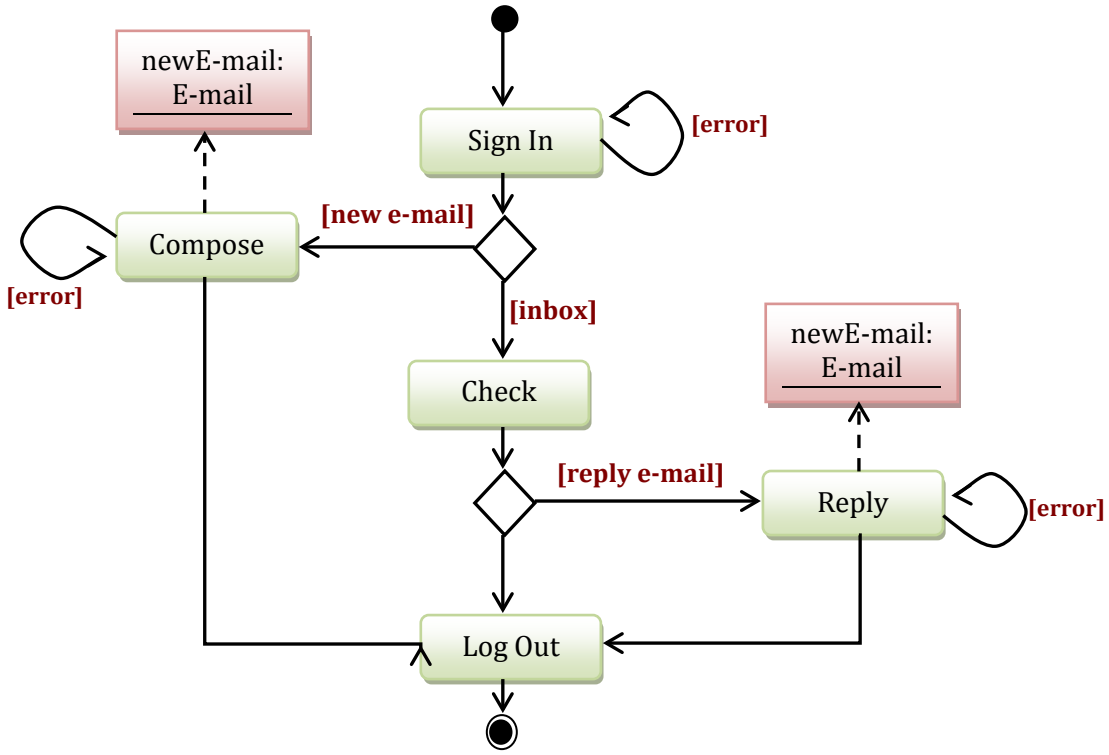



**Figure 4.29.** ***An example of an activity diagram***

In the figure above there is an example of a use case 'write an e-mail'. This particular process of the use case can be modelled in some different ways and the figure gives only one of them. The overall idea is for one to 'Sign In' to the system, then to choose whether to 'Compose' an e-mail immediately or to go to inbox and choose a particular e-mail and then to 'Reply' to it. All the ends of this process go to 'Log out' of the system. Labels on the arrows indicate why a particular flow can be done. The labels are not mandatory if the flow that is done is self-explanatory. What is more, in the given diagram there are two object symbols given. The thing is that by 'Composing' or 'Replying' one creates a new e-mail object.

The idea of modelling system's behaviour with this particular method can actually take immediately after information modelling. This is because the aforementioned suggestion that these activity diagrams can be modelled out of use cases. If to choose this approach, the navigation can be modelled exactly following the steps of the activity diagram. Nevertheless, all the models of the system should be compatible, so at least navigation should reflect integration points taken from this activity diagram.

## PROCESS MODELLING, SOLUTION 2

This second method for modelling the system's behaviour, as opposed to the first one, is suggested to be made after the navigation has been modelled, especially if one chose a navigation model that has to do with processes or tasks relevant to the particular system. And having in mind that all the models in the system have to be compatible, making this model after navigation makes this task easier.

Anyway, the second method proposed by [2] is based on UWE (UML- based web engineering), that we have already heard about in the previous parts of this thesis. This method proposes to

build so called *process flow model*. Building it consists of two parts: making a *structural view* and a *behavioural view*.

The structural view of the system is all about capturing process related information comprising structure and behaviour. This is done by analyzing the conceptual model that was presented in the information modelling part. In this schema classes are enhanced with a stereotype <<process class>>. The idea is to capture one process in one process class. Let's analyze an example with regards to the stated ideas; process structural view for the running example can be found in the following figure 4.30.

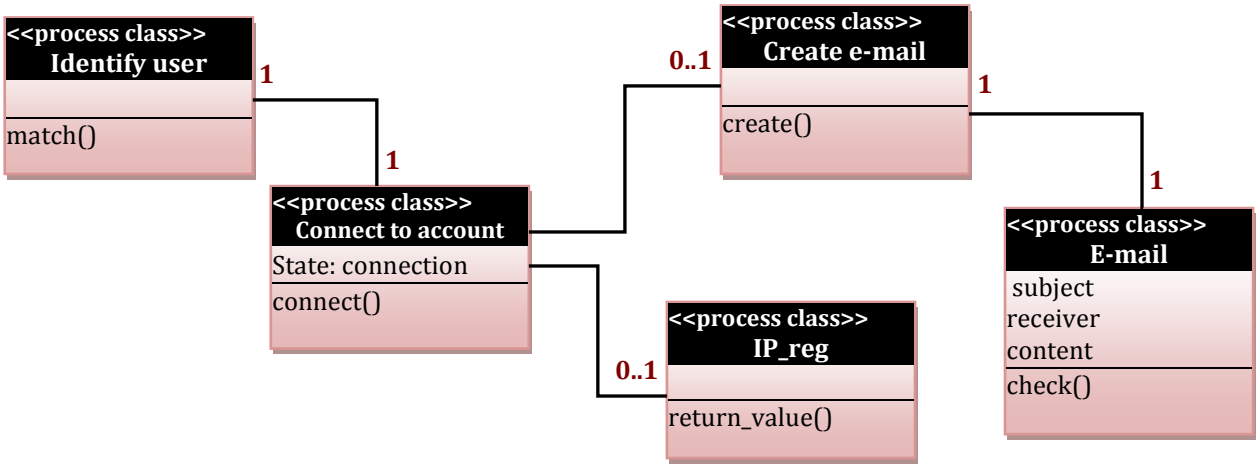



**Figure 4.30.** ***An example of a process structural view***

In the diagram of a process structural view there are four main process classes. Once again it is worth mentioning that this model is one example out of many possible even for the same system that is being modelled. Now, as it can be seen, each of these process classes represent one process: 'Identify user' is responsible for acknowledging user's credentials, 'Connect to account' is responsible for accessing a particular account, 'Create e-mail' is responsible for creating a new e-mail object, and 'IP_reg' is responsible for returning the last registered IP value. The cardinality properties depict whether a link between given process classes exist or not during the runtime. If there is a need to introduce states of a certain process, *process states* can be expressed as attributes in the process classes. As an example, 'Connect to account' process class contains a state attribute 'connection' in order to be sure if the connection to the account on the server has been made. The last thing to mention here is that this kind of diagram is capable to capture and data validation activities. As an example, process class 'E-mail' is responsible for validating (by applying *check()* activity) data structures which are explicitly given as attributes in this process class.

*Process flow model*, for this matter, is made by matching the structural view into an activity diagram, which is proposed by UML. This approach is detailed discussed in the previous part of the behaviour modelling, so it won't be discussed here again. It just worth mentioning, that (as before) one activity node presents one action or set of actions; but that set of actions (sub-activity) has to be modelled in a different activity diagram if one needs such a model (figure 4.31).

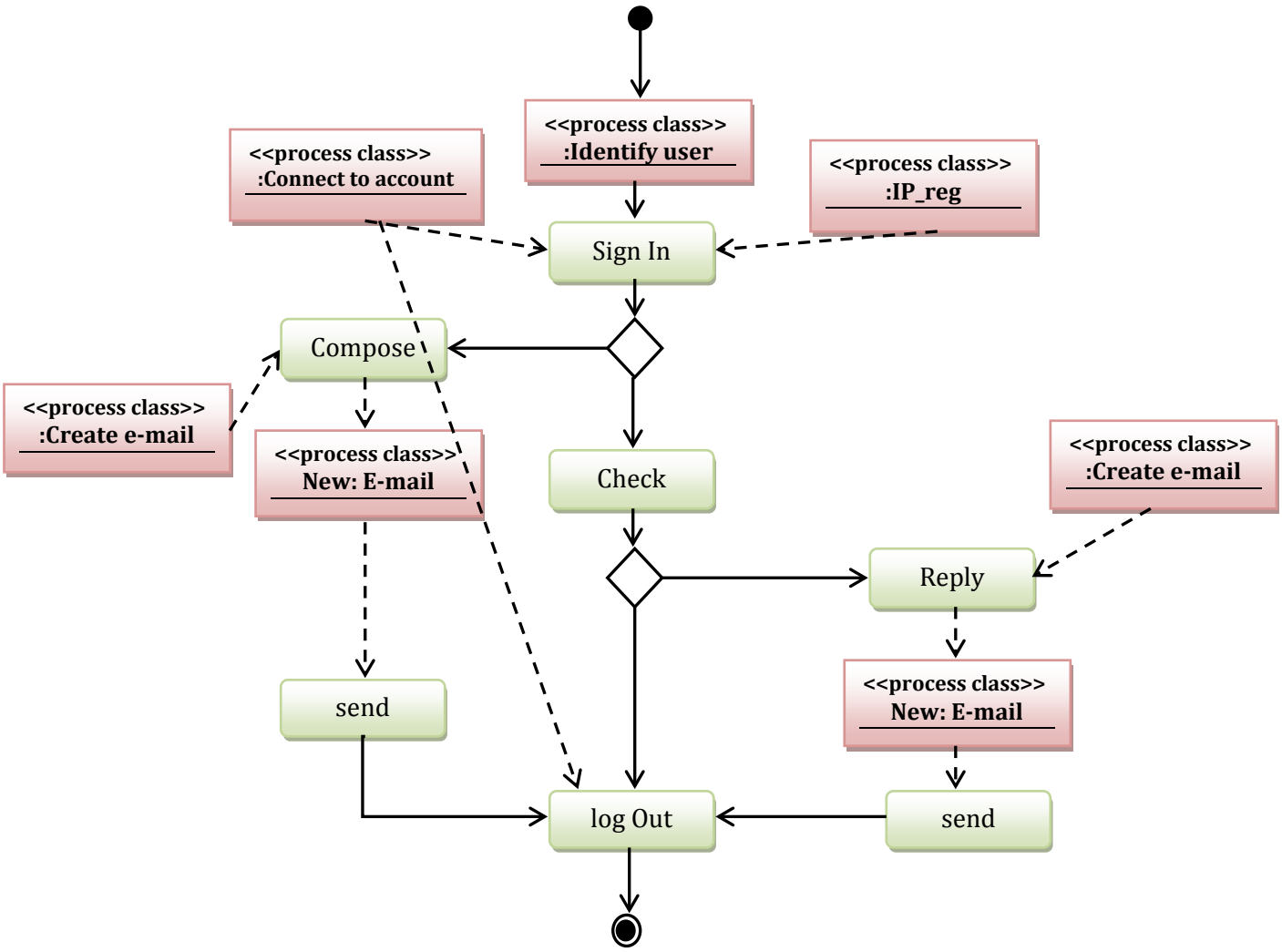



**Figure 4.31.** ***An example of an activity diagram (process flow model)***

The idea of such a schema (a process flow model) is to explicitly show how modelled activities are related with information structures – process classes. Other than just mentioned process classes, the activity schema itself is very similar as we had before. Nevertheless, there are some details to mention that are relevant only for process flow models. First of all, in this schema sub-activities (sets of actions) that can be modelled in other activity diagrams are the ones that have a name starting with a capital letter. Those actions that are done in one step (or *calls* as [2] gives) have names starting with non-capital letters. These usually are the ones where validation is made (as 'send' in the above example) or they can be those that change a process state (as 'log Out' makes one not connected to the account). But anyway, this is not a strict rule. The second thing to mention – it is easy to capture with this model situations where a process flow depends upon the results of operations of process information model; for that guard expressions can be used on the flow branches. This particular schema does not capture this dependability, though.

## DISCUSSION

In this part of the thesis there were two solutions presented for behaviour modelling in Web based information systems. The first solution is based on capturing the behaviour of the system in activity diagrams. These activity diagrams are very similar to what UML suggests but with

some adjustments. The overall idea is to make the diagram from the user's point of view and, what is more, the suggestion is to model in this way all the use cases. One might remember that use cases were discussed in the information modelling part. And if to have in mind the overall requirements processing, next step would be navigation modelling. This is not exactly a rule but a suggestion that might ease the workflow of requirements processing in general.

The second solution for behaviour modelling is about applying the UWE method. The overall behaviour modelling is also based on the approach of activity diagrams but the solution itself is done in two steps. The first step is to make a process flow model which includes capturing behaviour aspects in process classes. The second step is to incorporate relevant process classes into activity diagrams and in this way to make a structure which is called a process flow model. This second solution supports an idea that behaviour modelling is done after the navigation modelling. This is because of a fact that relevant process classes can be taken from a navigation model that UWE suggest. This method was given in navigation modelling part of this thesis.

## 4.5. PRESENTATION MODELLING

In this part of presentation modelling there is a method presented which suggests building abstract presentation diagrams and this method is presented by both [4] and [17]. The method is based on 'pattern' notion and expresses the requirements in Web presentation light. After stakeholders agree on abstract presentation diagram, actual Web pages can start being implemented. In addition to this, some ideas about storyboarding are given based on [3] with relation to the final presentation of the Web system.

What is more to the presentation, two more solutions of modelling the Web pages are given. The first solution for presentation modelling is suggested by [2] and it is a follow-up to the UWE approach, not that it is so relevant at this point. The overall idea is to group elements of the pages and in this way to give a sense of actual user interface. The second solution is presented in Hennicker, R. and Koch, N. *Modeling the User Interface of Web Applications with UML*[24] and it is all about showing the presentation in the light of navigation, which is to illustrate how the windows and parts of them are related and what the exact relationships are. What is more, the method is supported with sequencing of interaction with those windows and parts of them.

### ABSTRACT PRESENTATION DIAGRAM

The idea of an abstract presentation diagram is presented in [4] and [17]. The overall thing that is going on here is borrowed from OO-H methods and it deals with putting requirements in such a structure which is one step away from Web page implementation.

The way of constructing an abstract presentation diagram involves templates defining by structures as document type definitions (DTDs) or XML schemas. Nevertheless, this can be done

all the other way around; that is, the aforementioned structures can be formed from abstract presentation diagrams. What is more, there is a set of rules of how to transform navigation access diagrams to these presentation diagrams but in reality nothing is so totally straightforward. Generated default diagrams usually need further work as refinement to actually be useful for implementation. Anyway, the most important thing here to us is to actually figure out how exactly this technique works, and why it is important regardless of what kind of technical issues lay behind them.

Now, there are five templates used for modelling an abstract presentation diagram. These are summarized in the table 4.22 below.

**Table 4.22.** ***Modelling templates***

| Template | Purpose |
|---|---|
| **tStruct** | Information that has to appear on the abstract page |
| **tStyle** | Instances of this template define features that are present in a certain page or in a collection of pages |
| **tForm** | Defines data items that are required from a user in order to interact with the system |
| **tFunction** | Captures client's functionality |
| **tWindow** | Defines simultaneous views that are available to the user of a certain page or a data structure |

An example of a presentation diagram containing those templates is given in the figure 4.32 below.

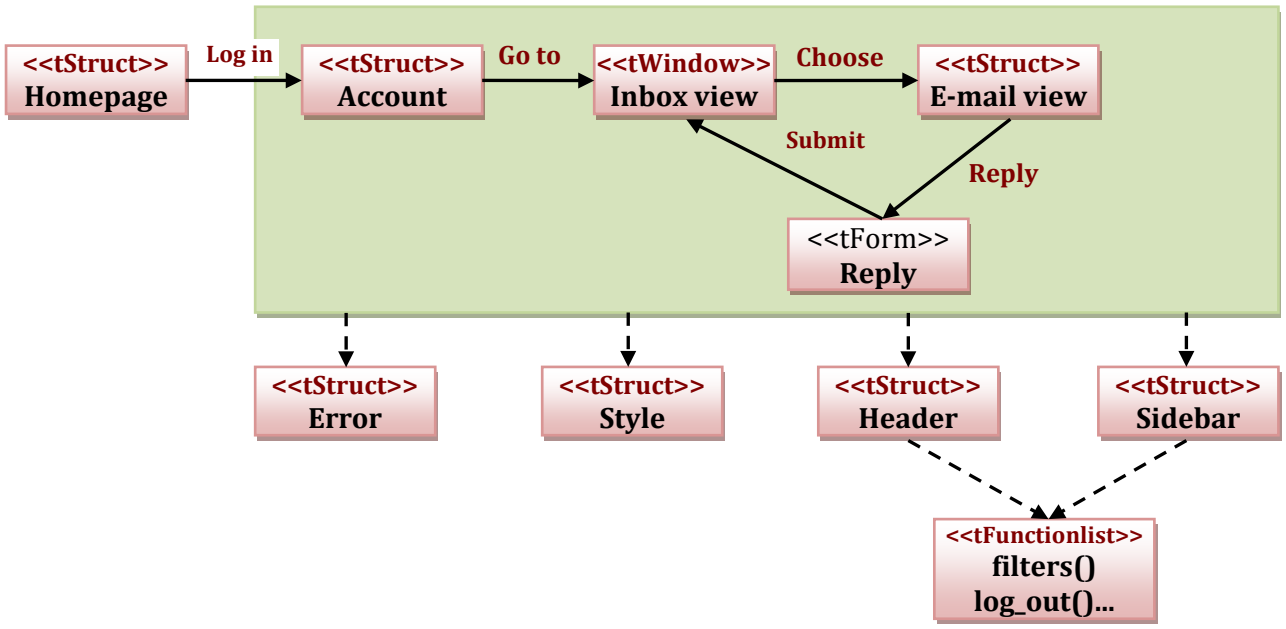



**Figure 4.32.** ***An example of a presentation diagram***

Here are some thoughts about the symbols used in the above figure:

**Table 4.23.** ***Symbols of the presentation diagram***

| Symbol | Purpose |
|---|---|
| **Normal arrow** | Show transitions between patterns |
| **Dashed arrow** | Shows relations among patterns |
| **Little rectangle** | Encapsulates a particular element of a pattern |
| **Big rectangle** | Groups some elements in order to optimize the expressiveness of the schema |

The given abstract presentation diagram shows that from Homepage one can go to the 'account', then to go into an 'inbox' which is provided by a certain view (there are many ways for a user to group or filter the content of the inbox). Next thing is that one can choose a particular 'e-mail', where a reply can be done using a form which has to be filled and submitted. Now, all of these elements except for the very Homepage itself have something in common. First of all, error pages, style of the pages, the header, and the sidebar are all the same for the elements. And what is more - the header and the sidebar are there to provide some functionality.

Overall, with this method one can capture and describe all the things that actually have to be in Web pages and in this way to predict how the actual Web site would look like. This diagram captures a sense of navigation, but there are many details that cannot be captured by any usual navigation models. These details are aforementioned style of Web pages, error pages, headers, footers, and sidebars. Nevertheless, the presentation diagram can capture whatever is relevant and not explicitly mentioned here, as plug-ins, scripts, etc.

## STORYBOARDING

The idea of storyboarding is not exactly a method to get a diagram or a schema and in this way to show system's characteristics, instead it is more a theoretical technique. The idea is presented in [7].

The hierarchy of some main concepts is presented in the following figure 4.33:

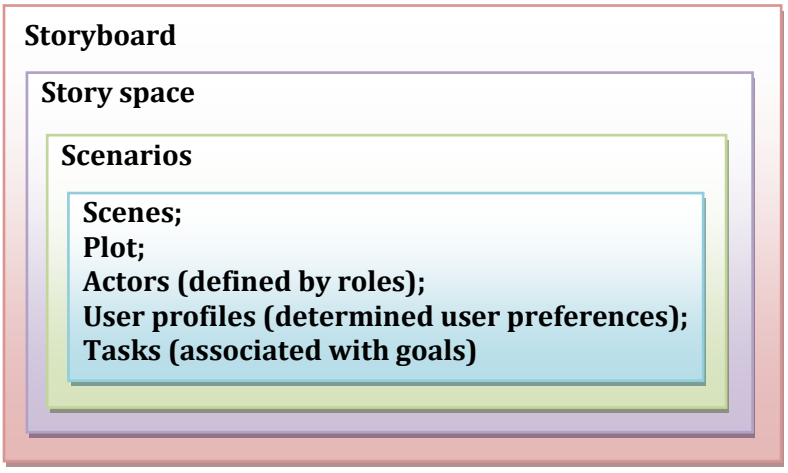



**Figure 4.33.** ***Concepts and hierarchy***

Now the figure above says that a Storyboard consists of Story Spaces, which consist of Scenarios which are usually defined by scenes, plots, actors, user profiles, and tasks. Here we have many familiar concepts. About scenarios we talked in the beginning of the thesis because the requirements gathering involve defining scenarios. We talked about actors in information modelling and extensively talked about tasks in operation modelling. Because of this reason, some of the ideas will remind what we are already familiar with; others, for that matter, will introduce some new approaches that are relevant for presentation modelling.

First of all when dealing with raw requirements one has to take in mind many things. Now, [7] suggests that semantics of a storyboard is defined by a set of stories it contains. In this case, a story is a path through a story space and is associated with some particular tasks in the system, or some particular roles of the actors. The main question is here: what do the stories mean to the users. We already talked in the beginning that requirements (in a form of scenarios) taken from users are not *real* requirements. Real ones have to be extracted from the raw ones by actually understanding what a stakeholder meant when providing a certain story. The meaning is all about the intention of what one's purposes are. To solve this matter a technique of so called *life cases* can be used. Life cases can be understood as observations of different aspects of users' skills and experiences, significance of time and place, etc.

Having discussed all the above issues we come to discuss things that are related to the presentation. If we are talking about a system where actors have different roles, it is important to perform so called *user modelling*. The idea is all about specifying user profiles and portfolios in order to recognize which tasks are exactly relevant to which roles. In the running example we have already established that we have three different roles: a student, a teacher, and an administrator. The separation is made because of specifications of specific work approaches, so this can be called as being a *work profile*. There are two more profiles that [7] suggests: education profile, and personal profile.

Now we came where a notion of a screenography has to be introduced. For this matter let's borrow a definition from [7]: 'Screenography aims at an individualized, decorated playout in consideration of user profiles and portfolios, provider's aims, context, equipment and storyline process. '.

One of the most important aspects to discuss here is an atmosphere. It is formed not only by colour palette, but also by shapes, material, and illumination. There are many things to consider for presentation designers, and it is a totally different science here to enforce emotions and moods through colours, and shapes. For the sake of thoroughness, it is worth mentioning that there are many different types of ambience considered in a Web system such as romantic (giving romance and passion), powerful (giving a sense of drama), balanced (giving a sense of harmony), etc. All this is a part of presentation but the way things are done for this matter cannot be modelled in a conceptual sense.

Nevertheless, there are some related concepts that can be briefly discussed. Web pages usually contain so called *functional elements*. These are icons for functional or navigational purposes. What is more, there are *visual elements*, having in mind text, pictures, structures, not to mention colours, etc. All this stuff together is called a *screen layout*.

The screenography we were talking about adopts some concepts from psychology: visual communication, visual cognition, and visual design. The importance of these is summarized in the following table 4.24.

**Table 4.24.** ***Psychological aspects***

| Principle | Purpose | |
|---|---|---|
| **Visual communication** | It is a precondition for an interaction support and consists of vision, cognition, and processing and memorizing characteristics: | |
| | **Vision** | Captures the fact that a user is depending on his or hers physical and psychological properties |
| | **Cognition** | Psychological and physiological skills and abilities |
| | **Processing and memorizing characteristics** | A psychological ability to read, integrate and reason about the content that a certain page provides; and to memorize part of the content |
| **Visual cognition** | One has to have in mind that users are limited with respect to time, attention, scope, and task portfolio | |
| **Visual design** | It is related to science of many different principles such as reading direction principle, similarity principle, etc. | |

The ideas about storyboarding and screenography are not something that can be modelled but something that models have to be aware of and take into account. It wasn't something that is crucial for this thesis but things discussed here are overall relevant and the presentation modelling part wouldn't be as complete without them.

## PRESENTATION MODEL, SOLUTION 1

This presentation method is presented by [2] and is based on UWE approach. The idea is to build a physical representation on the logic model. The logic model at this case is all the conceptual models that have been done so far up to this point.

The method presents two different views of the system: a *structural view* and a *user interface* (UI). The structural view shows the structure of the presentation space, and user interface shows the details of the design of the actual Web pages.

Talking about the structural view, the idea is to show how the presentation space is divided; that is, how the elements of presentation are displayed at the same space but not necessarily at the same time and how those elements of presentation can be grouped. There are some stereotypes that are used to make sense of this model. These are summarized in the following table 4.25.

**Table 4.25.** ***Stereotypes of the model***

| Stereotype | Purpose |
|---|---|
| **<<location group>>** | Location is the central concept around which structuring takes place; location group is used to model a set of pages (so called presentation sub-structure) |

| | |
|---|---|
| **<<location alternative>>** | These are to model presentation alternatives among <<location>> classes |
| **<<presentation class>>** | It is all about presenting logical page fragments and is composed of user interface elements (that are presented in the Web page) |

We have talked about the UWE method in the navigation modelling, and in behaviour modelling. Now, if one would choose to make all the modelling parts consistent with UWE, then <<presentation class>> element would have to be related with only one <<navigation class>> (from navigation modelling) and with only one <<process class>> (from behaviour modelling). The presentation structure of the running example is given in the following figure 4.34.

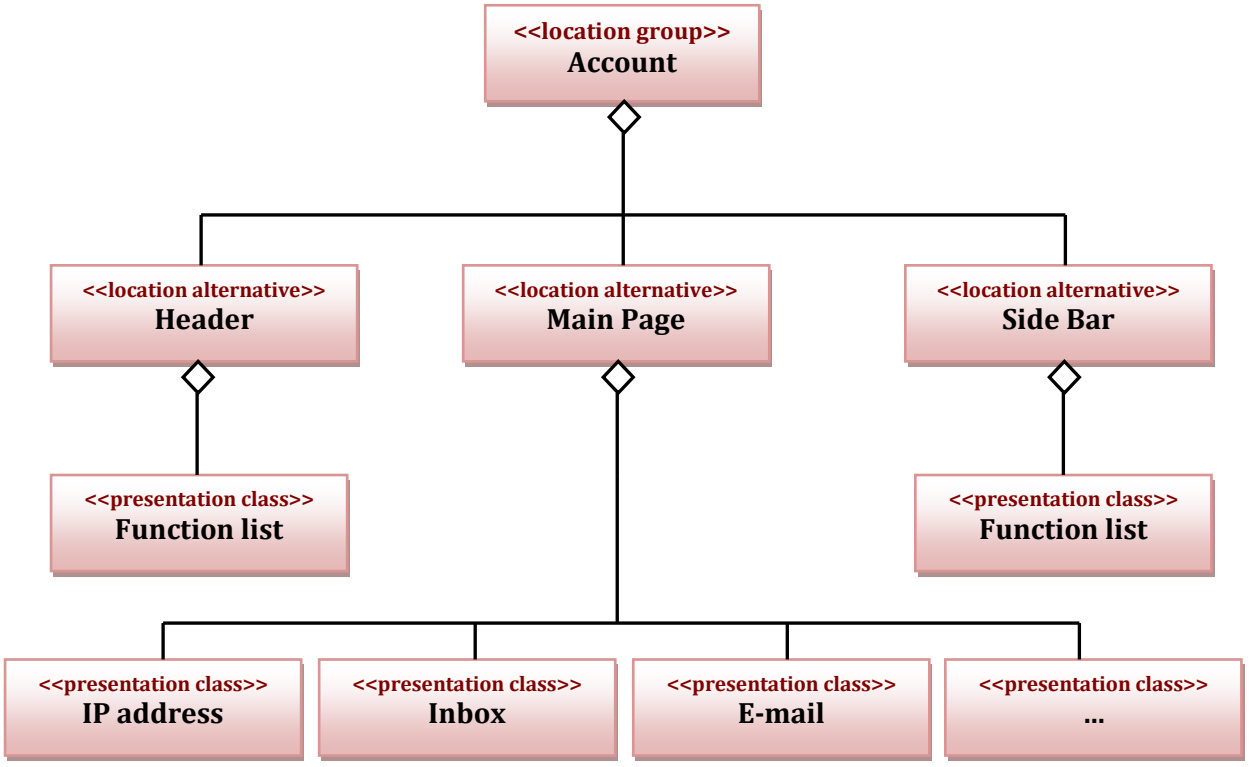



**Figure 4.34.** ***An example of a presentation structure***

The figure above shows a presentation structure for the E-mail Account. As the account basically is the key concept around which everything evolves; 'Account' gets to be stereotyped as <<location group>> (although the name itself does not have a semantic meaning, it is important what it represents). Anyway, the account has three <<location alternatives>>: 'Header' (where we can find a list of functions of what can be done in this particular E-mail Account), then 'Side Bar' (having similar list of functions as the Header), and finally the 'Main Page' (or possibly the main frame; it is hard to name exactly, as the means of actual implementation are not captured by the method). So, the 'Main Page' is responsible for displaying information about 'IP address', show 'Inbox', reveal 'E-mail's' content, and many other things to be added if to make this model very thorough.

Having done this presentation structure, every <<presentation class>> together with other elements that compose the upper levels of the structure (in other words, together with relevant

<<location alternatives>> and, of course, <<location groups>>) go together to form a user interface view. As an example, the following figure 4.35 contains the user interface view of the 'Inbox'.

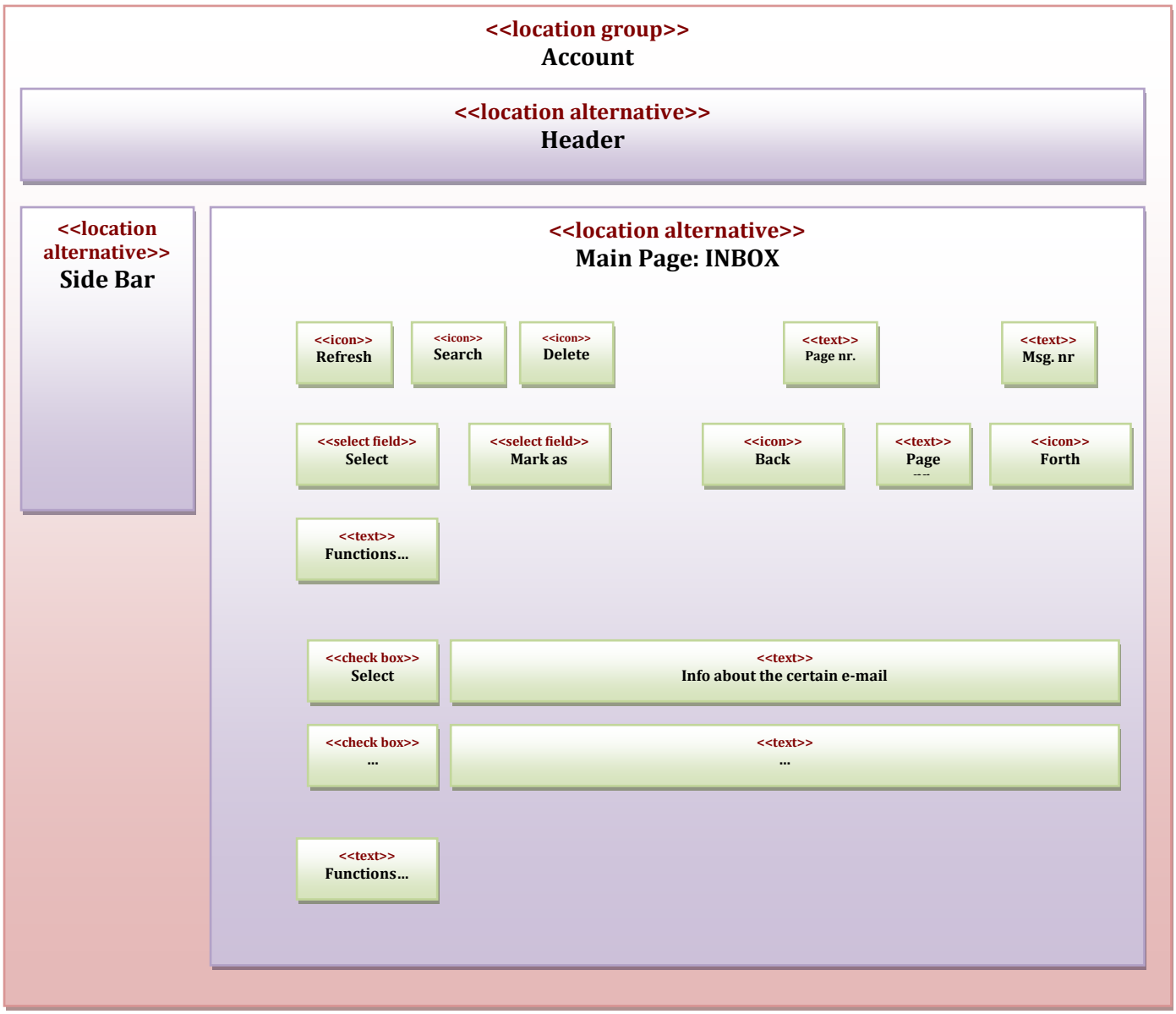



**Figure 4.35.** ***A user interface view***

The figure above captures the user interface view of the 'Inbox'. It seems that this model is very detailed but the truth is, if to capture all the Web page elements including one, this should be even more detailed and, the fact is, modelling the user interface is a hard thing. To do this is even harder with respect to the fact that it is impossible to know all the functions that will be needed in advance. Nevertheless, it has to be compatible with all the models that have been modelled before. So this example encaptures all the structures of the Web page: the Header, the Side Bar, and the main part of the page. One can see that there are elements representing information about the list of e-mails together with elements allowing to work with these elements. So this kind of a model can have elements whatever HTML or other Web page building tools have to offer.

This is the last idea that suggests some kind of a presentation model. This idea is taken from [24] which discuss more approaches than just presentation modelling; it includes thoughts also about user interface and navigation. Nevertheless, in this thesis we already discussed more or less everything that [24] has to offer in general, only the presentation model gives some new approach and is interesting to be seen here. Other than methods of presentations discussed so far, this particular idea supports capturing very detailed aspects of Web pages of the Web system.

This model uses stereotyped classes and stereotyped associations to capture the presentation very precisely. As it is given in [24], these stereotypes are summarized in the table 4.26 below.

**Table 4.26.** ***Stereotypes of the model***

| Stereotyped class | Purpose |
|---|---|
| **<<target>>** | It is an abstract class which generalizes a window together with frames; these are both compartments of a Web page which is displayed |
| **<<window>>** | It is an area of user interface; there are objects displayed; the window itself can be maximized, minimized, resized, reduced; it can include scrollbars (vertical and horizontal) |
| **<<frameset>>** | It is a modelling element used to define multiple visualization areas within a window; a frameset can be divided into lower level elements called frames; framesets can be nested |
| **<<frame>>** | It is a part of a frameset; it is where certain content is displayed. |
| **<<client page>>** | It is a Web page on the client's side |
| **<<server page>>** | It is a Web page on the server's side |
| **<<client script>>** | It is a script embedded in the client's side |
| **<<server script>>** | It is a script embedded in the server's side |
| **Stereotyped association** | Purpose |
| **<<builds>>** | It is used when a client's page is generated in the server side |
| **<<redirect>>** | It is used when another Web page has to be displayed instead of a selected one (or intended one) |
| **<<submit>>** | This is an association which allows to suggest that out of the data in a certain *form* a new object (or attributes of an object) is created in the system |
| **<<links>>** | It points from an anchor to a Web page (individual connection between two objects) |
| **<<displays>>** | It specifies that in the target a Web page is displayed (it can be a frame or a window) |
| **<<targets>>** | It points from an anchor to a target (a target is used to display the object linked by the anchor). |

These stereotypes that are summarized in this table are general ones and usually a subset of those is used when building an actual model. What is more, other than stereotyped associations, there can be used aggregation and specialization relationships among the stereotyped classes. Of course, if there is a need, one can use other stereotypes in case there are types of classes or

relationships that are not captured by this table. Especially, if a Web based system's interface is implemented not only in HTML or similar structures.

Now, as we have some elements to work with, let's try to build a presentation model for our running example about the Web mail system (figure 4.36).

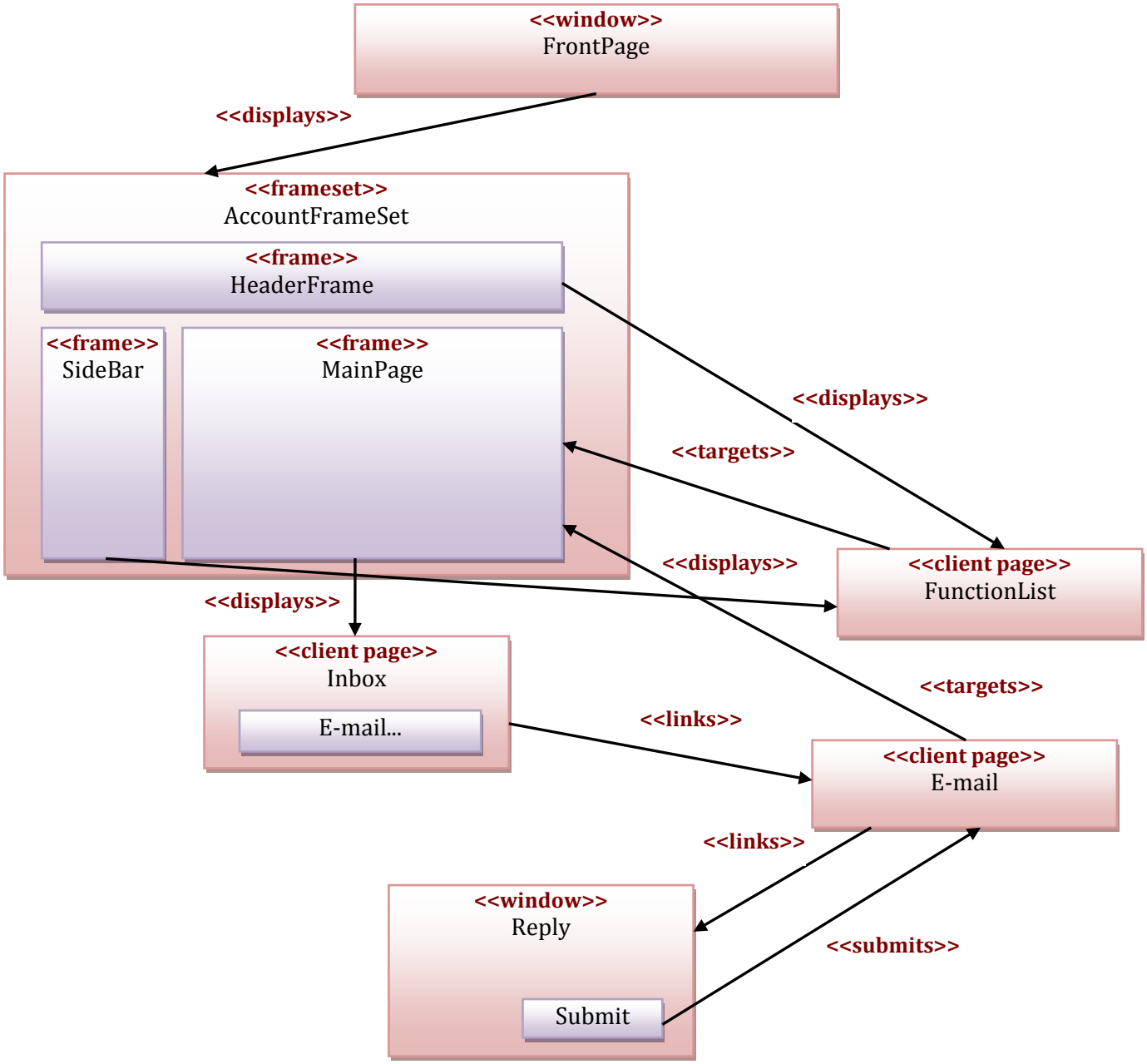



**Figure 4.36.** ***An example of a presentation model***

The presentation model above captures only a part of the presentational view that should be in the overall model. This is all right, since to model the full view one would rather make many models that suggest the correct presentation of the system, then just making a big one that might not make too much sense. Of course, if a system is quite simple and small, one model might also be the best solution.

Now, the given presentation model first of all suggests that there is one 'Front Page' from where one can access the 'Account' (this is indicated by 'Account Frame Set') and that 'Front Page'

actually is used for 'log in' purposes to the system. There are two frames in the 'Account Frame Set': 'Header Frame' and 'Side Bar'. These are used for displaying a client page for 'Function List', whereas the result of selected functions is displayed in the 'Main Page' (that the result will actually be displayed in the 'Main Page' indicates the associated relationship <<targets>>). If such a function selected, the 'Main Page' displays 'Inbox', where one would be able to choose a certain 'E-mail', which again would be displayed in the 'Main Page'. Now, if one would like to reply to an 'E-mail', this would give some sort of a form where input data could be submitted, but the form would be given in a separate window, therefore, association relation <<target>> is not used in this case.

So, basically, this kind of a model captures the presentation with the sense of navigation. What is more, it requires exceptionally that all elements that are used would be modelled and would capture the relationships among them. Other than just having this presentation view, one could form so called presentation flow if that helps to capture the dynamics of the presentation better. The presentation flow is modelled in a sequence diagram borrowed from UML and we have already discussed them while talking about navigation modelling. We will not discuss the technique once more but just analyze an example (figure 4.37). And just to note, in presentation modelling objects of the sequence diagram are windows or frames.

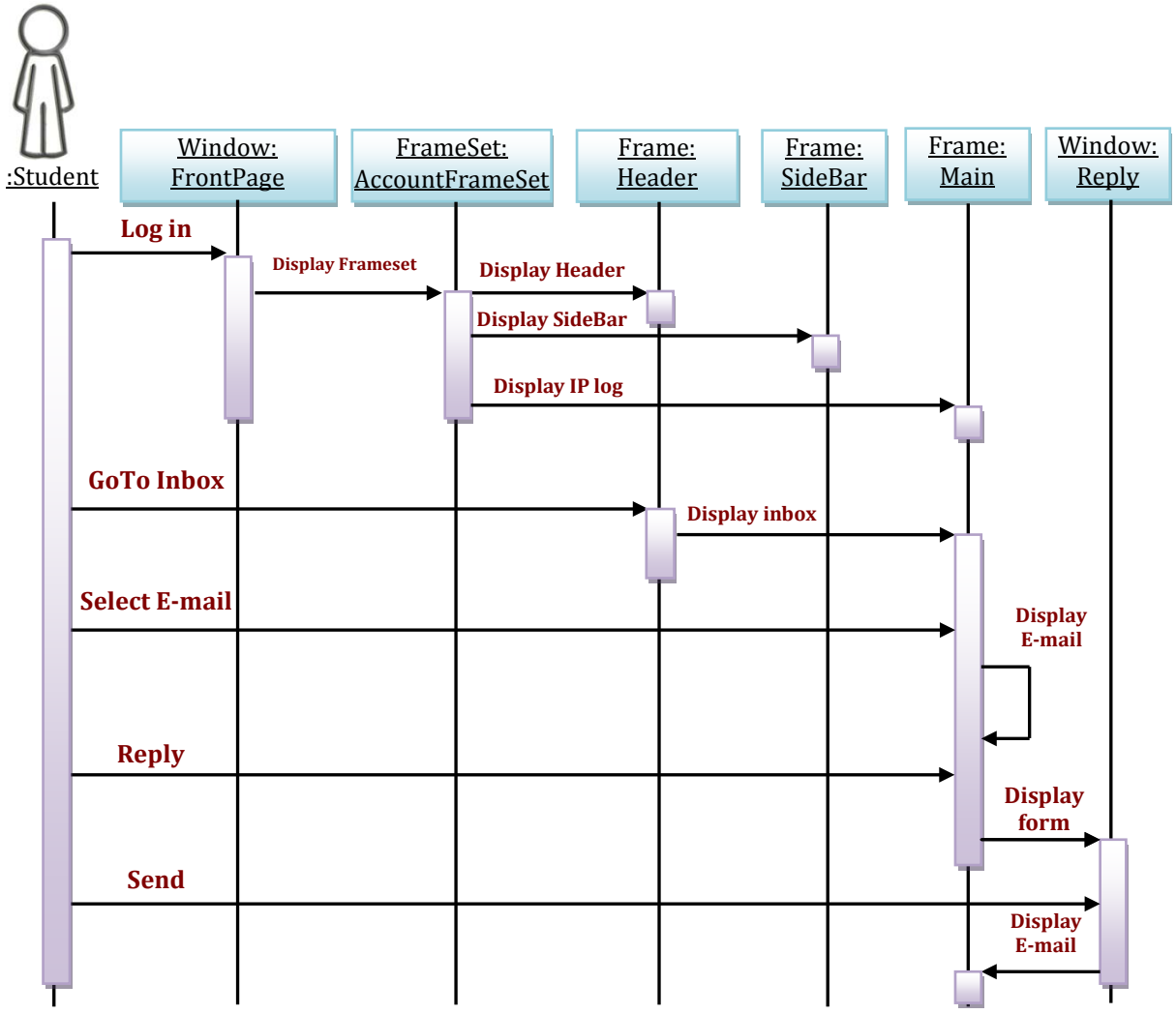



**Figure 4.37.** ***Sequence diagram of a presentation model***

This way of presenting the presentation flow supports the model given before in a way that it shows the sequence in which the relevant interactions between the user and within the windows and parts of the windows are done. As it was just said, it involves the user, so that he or she could clearly see how things are done and more - *where* they are done.

## DISCUSSION

In this part there were four ideas discussed with regards to presentation modelling. There was an approach presented which talks about building abstract presentation diagrams. The overall idea of those is that diagrams can capture data structures represented in different Web pages, multiple views of the pages, and provide some navigational aspects. But what is more, one can model with these diagrams things as error pages, style, functional aspects and all the relevant parts of a page which not necessarily can be modelled by any other type of modelling technique.

The second part of the presentation modelling was more theoretical one. The purpose was to remind some basic things about requirements and relate them to presentation modelling. But this time these requirements talk only about the impact of the presentation towards a particular user and emphasize the importance of ambiance and psychological aspects. Say what you want, but if a designer builds a system for managing funeral home, the background should never with a picture of pink bunnies. And as there is no practical way to model these things, at least this kind of requirements must not be forgotten and taken seriously as any other kind.

The last two ideas are going more into details of giving some presentation models. The solution number one is about collecting all the elements that exist in the system together to make sense out of them and present user interface views for different parts of the system. The second solution analyzes the windows and parts of windows of the system together with relationships among them. For that stereotyped classes and stereotyped associations are used. What is more, this solution additionally allows to model interactions of the user with the system, thus letting to feel the actual flow of actions with regards to the presentation.

# 5. SPECIFIC REQUIREMENTS FOR MODELLING

Other than information, navigation, operation, behaviour, and presentation modelling, Web based information systems have some more aspects to think about. Concepts such as business modelling, services, and transactions are closely related to the Web based systems. It is not that every Web site has to do something with this but especially commerce Web systems are related to business modelling aspects; and service systems with service modelling aspects. For this matter, it is important to give a little attention for these issues.

## 5.1. REQUIREMENTS FOR COMMERCE SYSTEM MODELLING

Sometimes it is not enough to take into consideration only the fact that a Web system is being modelled. These systems can vary with respect to the purpose as well as with respect to technical means of implementation. Therefore, it is relevant to talk at least about some specific cases of Web systems. In this part of the thesis there are some thoughts concerning enterprise and business systems (that is basically from purpose perspective).

### ENTERPRISE MODELLING

The idea of enterprise modelling is not that it gives any new techniques, but instead it suggests some comfortable approaches when dealing with large-scale projects. Enterprise modelling, as it is summed up in Frank, U. *Multi-perspective Enterprise Modeling (MEMO) - Conceptual Framework and Modeling Languages*[10], is emphasizing the need for high-level abstractions and importance of multi-view approaches. The overall idea is to model different views of the company and allow uncomplicated interactions between them. Suggestions apply here to us in a sense that there are increasing numbers of companies that exist mostly in the Web space and mostly commercial systems, for that matter, as freely accessible statistical data suggests. Overall ideas relate to requirements processing and should help in putting thoughts together when it comes to conceptual modelling of similarly big systems.

As we are talking about requirements, the first thing to remember is that communication usually takes place between stakeholders that have very different professional backgrounds. Defining intuitive concepts is a key issue in avoiding misunderstandings. For that purpose, it might be useful to reuse existing concepts that have already proved themselves. On the one hand, a common understanding among all the stakeholders should be reached at least of the big picture on the system; on the other hand, detailed levels of abstractions should come into play when modelling more specific aspects, possibly including only relevant stakeholders concerned with that specific area. Therefore, handling different views of such a system is unavoidable and is efficient while working with enterprises.

All that has been said so far about enterprise modelling ideas do not add something new to the requirements processing as such. These only suggest that all the discussed techniques can be applied for different abstraction levels and different partial views of the system.

## BUSINESS MODELLING

The most usual technique when it comes to business modelling is business process modelling. And when a talk goes about process modelling, in many cases it is implied that it's business process modelling. This thesis talks about Web based information systems, and regardless of the type of the particular system (we discussed types in the very beginning of this thesis in part 3), one still uses the term of business process modelling. On the basic level these are captured by simple activity diagrams taken from UML [16], in the behaviour modelling part (4.4) and some navigational solutions (4.2); but it can be seen that ideas go beyond that. Every Web based information system can be described as being some sort of a business system, but not all of them are commercial ones, therefore, some parts of the requirements processing differ. The systems behaviour part (business process modelling) is similar to all kinds of Web systems. Nevertheless, commercial systems have many legal aspects that tag along that can be included in other types of models or they can be processed with different means. What is more, commercial systems have a notion of 'adding an actual value' which makes them different from all the other kinds of systems. These aspects are worth talking about separately.

## BUSINESS MODELLING VS. BUSINESS PROCESS MODELLING

The challenging idea about business modelling was presented by Gordijn, J., Akkermans, H., and Vliet, H. in *Business Modelling Is Not Process Modelling*[9]. It is based on the thought that a business model is not about a process itself; instead, it is about the value that is exchanged between actors. There are two different aspects distinguished: *business modelling* and *business process modelling*.

Business modelling is all about providing the design grounds for a system from a business point of view. The idea is to identify what is exchanged and who are those participants that are exchanging the value regarding to the business. The main concept here is *value*, which is an exchange between relevant stakeholders thus creating business in the first place. There is a list of questions that Distante, D. and Tilley, S. in *Conceptual Modeling of Web Application Transactions: Towards a Revised and Extended Version of the UWA Transaction Design Model*[8] suggests in order for this model to be representative enough:

1) Who are the value adding business actors involved?
2) What are the offerings of which actors to which other actors?
3) What are the elements of offerings?
4) What value-creating or adding activities are producing and consuming these offerings?
5) Which value-creating or adding activities are performed by which actors?

We will stop here in order to analyze an example. To understand the concepts better, it will be a simple example of a Web system where photos are sold-bought (the description of this system can be found in the appendix C). The idea is simple: professional photographers put nice pictures in the Web system where people can buy the ones they like by using a money payment service (such as PayPal, for example). The summary of the representation of the business model is in table 5.1 below.

**Table 5.1.** ***Business modelling***

| Question no. | Answer |
|---|---|
| 1 | Photographers (providing photos);<br>Users (providing money);<br>Business owner (providing the whole Web system);<br>Payment service administrator (PSA) |
| 2 | Business owner offers photographers an opportunity to post their work in order to sell it, what is more, owner offers a place for users to buy photos;<br>Photographers offer their photos;<br>Users offer money for the photos;<br>PSA offer correct and safe money transfer |
| 3 | Business owner offer a Web site (with some certain rules and policies);<br>Photographers offer photos of different kinds, different sizes, and quality;<br>Users offer money of different currencies;<br>PSA offer money transfer services (from users account to service's account; further to the relevant account of the photographer and the business owner) |
| 4 | Running server and administration;<br>Publishing photos;<br>Selling photos;<br>Purchasing photos;<br>Transferring money |
| 5 | The server handling is a responsibility of the owner;<br>Publishing and, therefore, selling is done by photographers;<br>Purchasing is done by users;<br>All the money transferring issues are done by PSA |

In this example we are talking about a very simple case of a Web business. Answers to the questions might seem straightforward and they appear to repeat the same information. But in case a Web system includes many stakeholders and handles not only selling part, but the process of production as well, these answers suddenly may appear complicated and take many pages. The idea is not so much about gathering information for functionality purposes; it is more for revealing true purposes of the system.

When considering businesses in general, there are rules to consider, and they are summed up in zur Muehlen, M., Indulska, M., and Kamp, G. *Business process and business rule modeling languages for compliance management: a representational analysis*[14]. Basically, a business rule is a declaration about organization's behaviour and information. There are two types of classification suggested: according to the *source* and *structure*.

Classification according to the source:

- *Mandates* are common published policies that have to be followed because it is considered under the law. Examples: payment of taxes, produce authentic products, etc.

- *Policies* are published declarations that concern only a particular organization, it is like internal companies rules. Examples: mission statements, budgets, etc.

- Guidelines are such organization's declarations that according to the circumstances may apply or not. Examples: methodologies and management styles.

Classification according to the structure:

- *Integrity* talks about particular internal rules or constraints, such as each project can have only one manager, or that a trial version of a project has to be done within one month.

- *Derivation* is conditions that result in conclusions, such as if a customer buys five photos online, one can choose one more for free. This means if one buys five photos, the conclusion is that one can choose one more and not pay for it.

- *Reaction* is defined by nouns: event, condition, action, alternative-action, post-condition. A customer buys many photos, and the amount of those photos is more than 20, then a 'thank-you' letter is sent by an ordinary mail (if that is shared) or by electronic mail system otherwise.

- *Production* is defined by: condition, and action. If a trial project is tested and contains no more errors, it is approved to be a final version.

- *Transformation* refers to a change of state. If a user wants to buy a photo, the amount of photos in a shopping basket can go only from 0 to 1, and not from 0 to -1.

There are a few more relevant aspects that we can borrow from the paper [14]. While the main idea of the work is to analyze business modelling languages, some concepts are relevant here for us as describing the business itself and are worth thinking about when expressing requirements. These concepts are *lawful event space* and *conceivable state space*.

Lawful event space is all about describing the set of events that is permissible in a system. This is basically all the collections of all the events that are legal with respect to the organization. Now the conceivable state space describes all the states of the system that can be assumed about the system but not necessarily explicitly stated. To separately discuss these aspects is relevant for risk management of the system because they might allow discovering unexpected contexts.

Differently from business modelling, business process modelling is all about the *way* the aforementioned *value* is exchanged. It includes setting a common approach for the work, process supported by systems, property analysis, etc. Nevertheless, there is a set of questions that [9] gives for this phase too:

1) Who are the actors involved in the operations?
2) Which operational activities can be distinguished?
3) Which activities are executed by which actors?
4) What are the inputs and outputs of activities?
5) What is the sequence of activities to be carried out for a specific case?

6) Which activities can be carried out in parallel for a specific case?

The answers to this set of questions are given below, in the table 5.2. The difference between business modelling and business process modelling can be seen by comparing these answers with the former ones.

**Table 5.2.** ***Business process modelling***

| Question no. | Answer |
|---|---|
| 1 | Photographers;<br>Registered, non-registered users;<br>Administrator |
| 2 | Registering as a user;<br>Registering as a provider (photographer);<br>Logging-in to administrator's account;<br>Logging-in to a provider's (photographer's) account;<br>Logging-in to a user's account;<br>Uploading a photo;<br>Providing a photo's description;<br>Setting a price to a photo;<br>Going through sorted photos;<br>Commenting (possibly rating) photos;<br>Signing out from an account;<br>Ban/delete users, providers;<br>Ban/delete photos...<br>... |
| 3 | Non-registered users (going through photos, registering as a user...)<br>Registered users (logging-in as a user, going through photos, commenting photos, signing out from an account...)<br>Non-registered providers (register as a provider (photographer)...)<br>Registered providers (logging-is as a provider, uploading a photo, providing a description, set a price, signing out from an account...)<br>Administrator (logging-in as administrator, ban/delete users, providers, ban/delete photos, signing out from an account...)<br>... |
| 4 | ...<br>Registering as a user (input: personal and access data; output: access to the system with buying, rating, commenting privileges)<br>Uploading a photo (input: a picture file; output: the picture is being published on the web page in order to be bought)<br>... |
| 5 | ...<br>(Specific case)<br>Go to registration page, fill-in registration form and submit it, wait for confirmation letter to the produced e-mail address, follow the confirmation link that can be found in the e-mail sent by the system, log-in to the system as a registered user. |

| | |
|---|---|
| 6 | ...<br>...<br>Having in mind the specific case defined above, one can go through photos in the photo gallery while waiting for the confirmation mail to arrive.<br>... |

This table, capturing business process modelling, basically sums-up detailed functional requirements. By the way, it is only a little part of everything that can be written about this (seemingly non-complicated) business system. Furthermore, when the questions talk about operations, we limit ourselves only to the user operations, as we already know from [3], there can be system's operations, which are not captured here. This only means that there are lots more processes going on inside the system itself.
For the matter of business process modelling, [14] suggests a classification for business processes. There are three main types considered:

- *Activity centered*. This is all about expressing processes as a network of tasks and activities.

- *Process object centered*. In this case processes are as a sequence of state changes of the process object.

- *Resource centered*. This type considers processes as a network of processing situations that interact with each other.

As we have already established the difference between the business modelling and the business process modelling, it is important to discuss the most relevant concepts with regards to that. These happen to be actor, value activity, value object, value port, value interface, and value exchange. These are summarized in the table 5.3 below.

**Table 5.3.** ***The most relevant concepts***

| Concept | Purpose |
|---|---|
| **Actor** | An actor in this case is understood as an independent economic, or even legal, entity. Actors do value activities and in this way add value. The actors can be either business actors or end-consumer actors.<br>-Business actors buy objects of value (considered input), they perform value activities and produce objects of even greater value (considered output).<br>-End-consumer actors create value by consuming an object of value. In addition to the resources one needs to consume that object of value, value is added if the utility that is assigned to consumption is higher to the utility associated with paid money for the value object. |
| **Value activity** | It is an activity that is performed in order to produce objects of value by adding value to other objects of value (that is to add value to inputs and thus produce outputs). Value activity must add value. |
| **Value object** | Actors and value activities exchange objects. These are called value objects. A value object can be a service, a thing or some consumer experience that has a value. Object's value is a relative idea; it is considered with respect to an actor. |

| | |
|---|---|
| **Value port** | Value ports are used for requesting and offering value objects from or to that object's environment. |
| **Value interface** | A value interface groups value ports, and shows the value objects that an actor might be willing to exchange with other value objects using those value ports. |
| **Value exchange** | A value exchange represents the trade of value objects between value ports. At least one input port and one output port has to be present in the value exchange. |

All the relationships between these concepts are given in the figure 5.1 below.

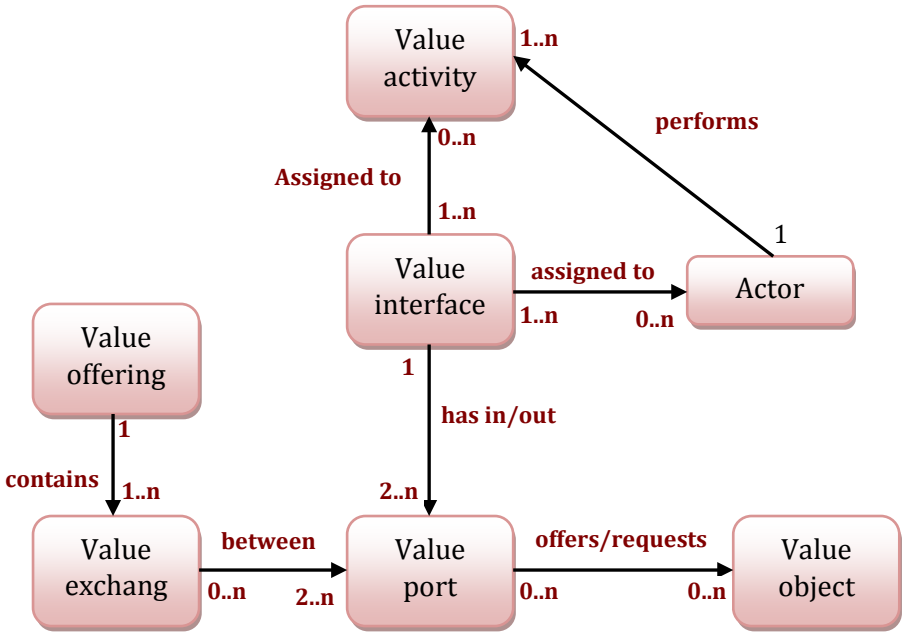



**Figure 5.1.** ***Relationships between value modelling concepts***

This is a general expression of the relationships between the business model concepts. The idea is that one value offering item can contain one or more value exchange aspects. Now, if there is indeed a value exchange it has a 'between' relationship with at least two value ports (one for input and one for output). If there is a value port – it can offer or request none or many value objects. What is more, there has to be one value interface which groups at least two value ports (one for input and one for output). Other than that, a value interface can be assigned to a value activity and assigned to an actor, now an actor (if there is one) can perform one or more value activities. An illustration of the relationships of the concepts is given in the figure 5.2 below.

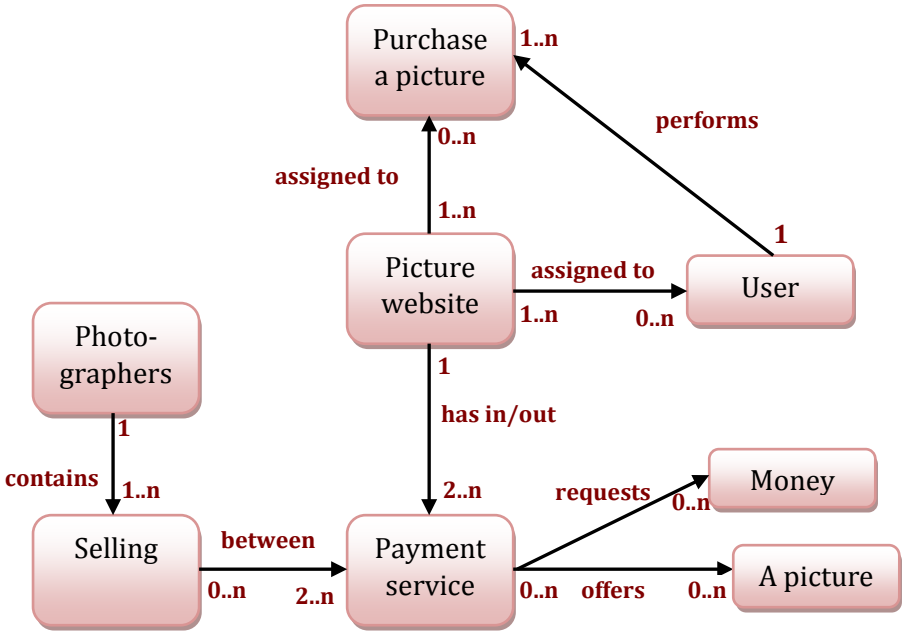



**Figure 5.2.** *An illustration of relationships between value modelling concepts*

In the above there is an example of how actual concepts can be applied to the general schema. Photographers (as value offering side) has selling in mind (as value exchange) that has a relationship with a payment service, with taking money as value and providing a confirmation that the money has been acquired (in this way giving a permission to let user download a picture). Now, the payment service (through the whole website) offers a picture as a value and requests money in return. A user has to have an account in the website and a value activity that is done is actual purchase of the picture. This means that a picture for a user has a greater value then the amount of money paid, otherwise one wouldn't buy a picture in the first place.

Having in mind all the given concepts, business modelling as well as process modelling can be done using activity diagrams. These were discussed in the part about system's behaviour modelling. The idea would be to use swimlanes for every kind of an actor that is participating in that business, and by having activities as value adding activities show the flow of the value exchange (figure 5.3). In this case we use boxes instead of swimlanes and some more specific symbols.

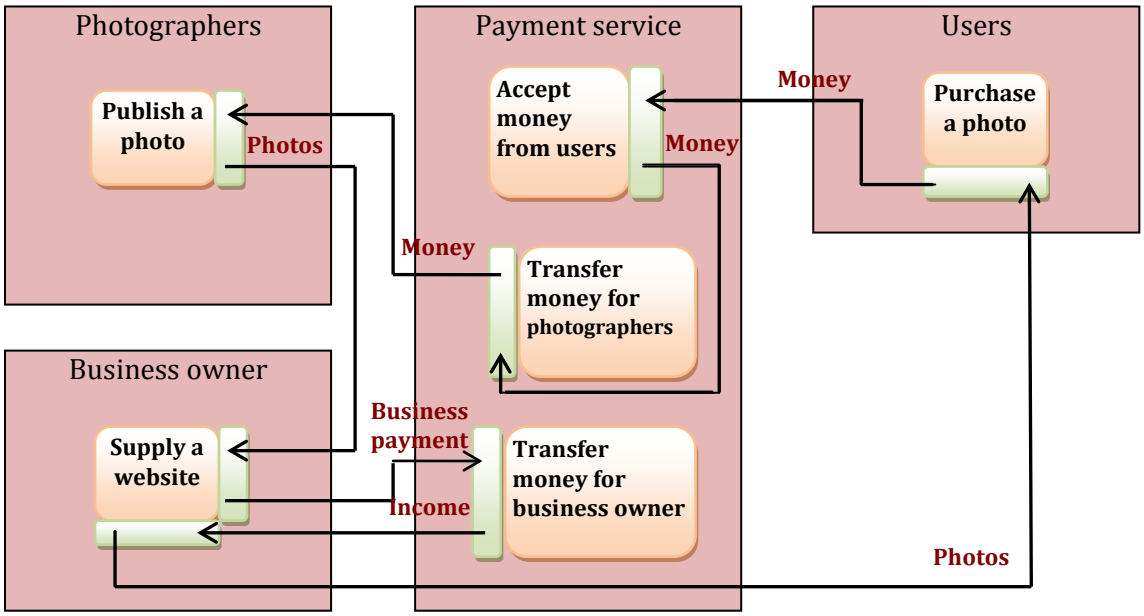



**Figure 5.3.** *A business model with value exchange*

In the figure above we have a business model with respect to value exchange. There are some symbols to discuss within this model. These are summarized in the table 5.4 below.

**Table 5.4. *Relevant concepts***

| Symbol | Purpose |
|---|---|
| **A square** | A square indicates an actor in the business |
| **A rounded-corner square text box** | In this way a value activity is expressed in the business model |
| **A rounded-corner square next to a rounded text box** | The rounded square represents a value interface and those points where an arrow comes or goes are value ports |
| **An arrow** | An arrow indicates value exchange and labels on those arrows describe value objects that are exchanged |

The figure shows a sense of how the value is exchanged between the actors participating in the business with respect to value exchange activities. The important thing is that every value activity has associated value interfaces with at least one input port and one output port. It does not matter from which perspective one would look at it, to get some value in business you have to give something in return.

DISCUSSION

The ideas that this part of the thesis propose to consider some suggestions in case there is a matter of enterprise modelling. In this case one has to draw attention to using commonly understood concepts, and multi-view modelling.

Business modelling (business process modelling having in mind) is usually associated with the idea of workflow modelling. Although, the most important thing is to realize that business modelling and business process are two very different things. The business modelling as it is, allows one to model the actual value exchange.

## 5.2. REQUIREMENTS FOR SERVICE-BASED SYSTEM MODELLING

Requirements processing can be important to show the perspective of technical implementation issues. This is due to two important aspects that are usually associated with Web systems, that is transactions and services.

As B. Benatallah, F. Casati, F. Toumani, R. Hamadi in *Conceptual Modeling of Web Service Conversations*[15] proposes, transactions are assurance for a user that some sum of operations can be aborted at any time without any negative or unexpected effects. This idea usually goes with the notion of services that will be discussed a bit later.

The thing with transactions is that in common practice they usually are treated as a somewhat kind of navigation. Transactions involve content navigation and also they are concerned with execution of activities governed by business process rules. [8] came up with a list of requirements for a proper transaction modelling:

1) All the activities that are involved or associated with any kind of a transaction has to be represented;
2) Execution flows have to be described; what is more, execution rules of each activity has to be defined;
3) It has to be specified which activities can be suspended/resumed;
4) There has to be described a way in which different types of users can participate in an execution;
5) It has to be specified how navigation and activity execution effect each other;
6) Content that supports a particular execution has to be defined; what is more, information objects that can be affected by an execution have to be identified;
7) Customisation conditions have to be discussed taking into consideration the context of the execution and the state of the transaction.

The idea of a transaction of a service operation can be defined with respect to whether or not an operation can be aborted at any time but without any effect. There might be more complex transactional models which would allow specifications for abortion of an operation, such as some specified timings for roll-backing or even fees for this kind of services.

Things about Web services are that they are not necessarily deployed by the same project team as the whole system. For this reason, specifications and descriptions of the services and description models have to be built. And as Benatallah, B., Casati, F., and Toumani, F. *Web Service Conversation Modeling: A Cornerstone for E-Business Automation*[25] suggests, these models have to have a definition of service properties that support: humans in understanding the service properties, clients in searching services based on these properties, and applications in automating the enforcement of the properties, much like transactional middleware supports transactional abstractions. The requirements of the way that transactions of services work in a system can be expressed by statecharts, an example is given in the figure 5.4 below.

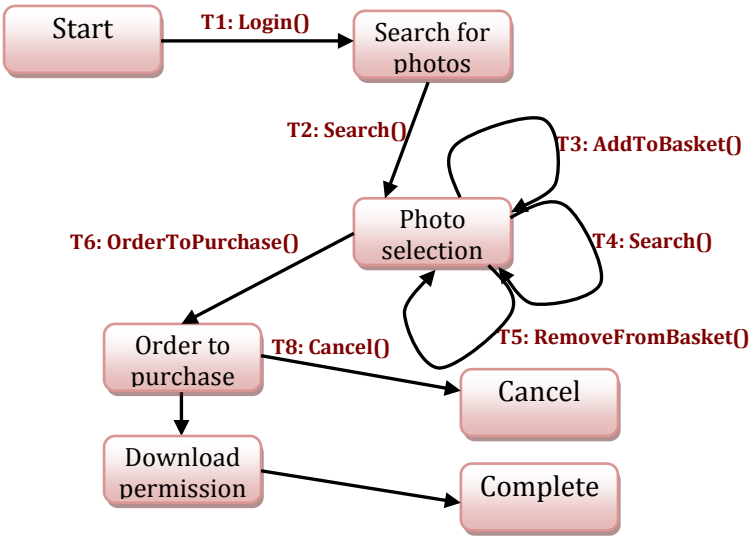



**Figure 5.4.** ***A statechart expressing a transaction***

Nevertheless, there are many concepts that are relevant for describing transactions of Web services. These are summarized in table below.

**Table 5.5.** ***Relevant concepts of Web services***

| Concepts | Descriptions |
|---|---|
| **Implicit and timed transitions** | Most transitions between states usually occur because of explicit operation invocations; it is possible for a state to change and without explicit operation invocation; what is more, transition can be done after a predetermined time interval |
| **Compensation** | Compensating operations are those that cancel each other's effect; usually this should be allowed within some certain period of time |
| **Resource locking** | Acquiring in advance some resources that a client might need |
| **Conditions and instance-specific properties** | This is due to a fact that before some operation can actually be executed, some certain conditions have to be verified first; or some aspects that concerns a specific operation can be instance specific (depending on that particular situation) |
| **Multi-state enabled operations** | Many operations can be executed in more than one state; and execution of an operation does not necessarily cause a state change |

The figure 5.4 above illustrates an example of a photo business system's behaviour. The idea is simple: we have divided the behaviour to some significant states and the schema reveals the basic transitions between the states. The transitions that are not labelled are either implicit or timed ones – not depending on user's actions. What is more, we can see an example of compensation – 'Cancel' operation. This operation compensates ordering for purchase, which means that it cancels it. Other than that, in the example one can also see that the execution of some operations does not necessarily change the state, as 'Photo Selection' state does not change when adding something in and out of the shopping basket.

In general, execution of a service transaction is made by exchanging a message with it. There are two categories of operations that involve one message, and there are two categories of operations involving two messages exchanged. These categories are summarized and explained in the table 5.6 below.

**Table 5.6.** ***Operations in transactions***

| Operations involving one message | |
|---|---|
| **One-way** | It is an operation initiated by the client of the service and consist of an input message |
| **Notification** | It is an operation initiated by the service and consists of an output message sent to the client |
| **Operations involving message exchange** | |
| **Request-response** | It is an operation initiated by the client and has one input message, which is followed by an output message from the service |
| **Solicit-response** | It is an operation initiated by the service and has one output message, which is followed by an input message as a response from the client |

Now paper [8] gives a fine definition of the Web transaction term: 'Web transaction indicates a sequence of activities performed by a user in order to carry out a specific task or fulfil a specific goal related to a business process by means of a Web application'.

The idea would be to say that there is no exceptional way to model transactions. Usually transaction modelling is associated with navigation modelling, but there are some things to know in advance in order to supplement those navigation models with the right information. Nevertheless, these have to be compatible with rules of the overall business processes.

If to think about a meta model for transactions, it should fulfil these requirements, as it is given in [8]:

- Represent activities that are involved in a certain Web transaction, but have to be compatible with a business process model;
- Describe the flow of the transaction execution together with associated rules;
- Specify which activities can be suspended and resumed;
- Describe the way a user participates in a particular execution;
- Describe information units that are supporting the execution and units that are affected because of execution;
- Give an explanation of how an activity will be customized with respect to the execution context.

## SERVICE MODELLING

The proposal that Quintero, R., Torres, V., Ruiz, M., and Pelechano, V. in *A conceptual modeling approach for the design of web applications based on services*[5] gives, is about modelling services. The exact ideas are: '(1) modelling external and internal services at conceptual level; (2) defining business processes as processes that are build from native and external service operations; (3) redefining some primitives from the navigational model to allow for the construction of web interfaces that provide access mechanisms for invoking services'. The problem space identified is content, navigation, and presentation. This approach was mentioned and in [11]. The overall purpose is to build such a navigational model, which would include not only internal and external data but functionality as well.

At this point [15] suggests a framework that has to take into consideration service properties that can support: (1) people actually understanding what service properties are; (2) clients that would look for services with certain properties; (3) applications enforcing certain properties.

Other than already analyzed activity diagrams for behaviour modelling, [18] suggests expressing workflow model with respect to services. The idea is to simply model a directed graph W=<N,E>, where N is a set of nodes and E is a set of edges. There are four types of nodes and four types of edges. Exact descriptions are summarized in the tables 5.7,5.8 below.

**Table 5.7.** ***Descriptions of nodes***

| Nodes | Descriptions |
|---|---|
| **Start node** | It is a unique node that indicates the beginning of the workflow, it has no predecessors (*symbol*: a rounded corner rectangle) |
| **End node** | It is a unique node that indicates the ending of the workflow, it has no successors (*symbol*: a rounded corner rectangle) |
| **Work nodes** | These perform actual activities in the workflow (*symbol*: a rectangle) |
| **Route nodes** | The responsibility of these nodes are only to evaluate rules; according to the evaluation the flow is directed (*symbol*: a rectangle with more rectangles as sides) |

**Table 5.8.** ***Descriptions of edges***

| Edges | Descriptions |
|---|---|
| **Forward edge** | Depicts normal workflow execution of forward direction |
| **Loop edge** | Depicts repeated execution loops |
| **Soft-sync edge** | This is a synchronization edge: this one signifies delay dependency between two nodes; means that a certain node can be executed if execution of the other one is already completed or cannot be triggered anymore (symbol: a dashed arrow) |
| **Strict-sync edge** | This is a synchronization edge: this one requires that completion of the execution of the other node is successful (symbol: a dashed arrow) |

An example of a workflow model with the described nodes and edges is given in the following figure 5.5.

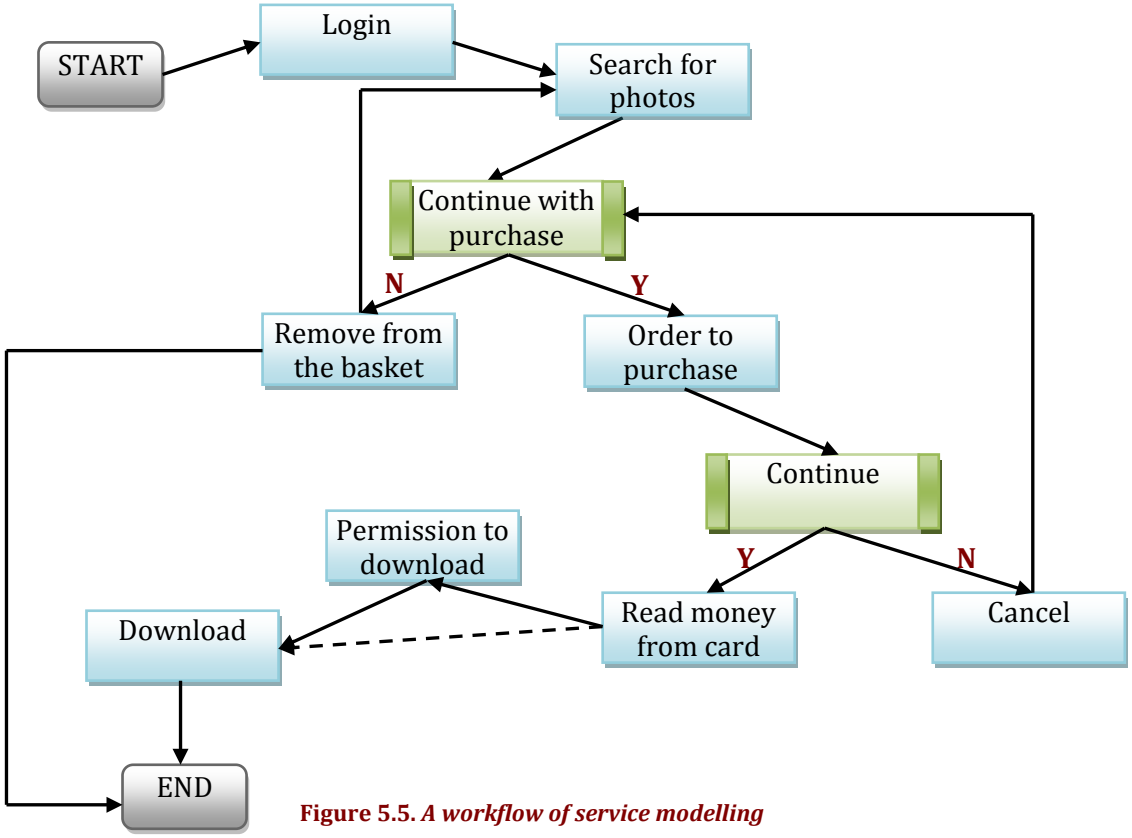



**Figure 5.5.** ***A workflow of service modelling***

The given example is quite self explanatory. We start the workflow by logging in to the system, and end it by downloading a purchased picture or simply by giving up the idea of purchasing something in general. As an example of a synchronization edge one can see it going from 'Read money from card' to 'Download'. This indicates that one can download a picture only when the money is paid, no matter what circumstances are.

Modelling of Web service execution differs from ordinary workflow modelling. As it was already mentioned, services may origin from different Web service providers, therefore, there might be sometimes very significant sematical gaps. And these are usually due to the underlying contexts. And these gaps have to be filled (table 5.9).

**Table 5.9.** ***Descriptions of contexts that help to fill in the gaps***

| Context | Purpose |
|---|---|
| **C-context** | Used for composite services: label, previous Web services, current Web services, next Web services, begin time, status per service instance, date |
| **W-context** | Web service provider: label, number of service instances allowed, number of service instances running, next service instance availability, status per service per composite service, date |
| **I-context** | Individual service instances: label, status, previous service instances, next service instances, regular actions, begin-time, end-time, reasons of failure, corrective actions, date |

This architecture works by taking into account two operations: consolidation during composition, reconsolidation during execution, as [18] suggests.

*Consolidation during composition*. After a Web service accepts an invitation of the participation in a composite service C-context as in (1, figure 5.6), a Web service instance together with an I-context is created as in (2, figure 5.6). Details of the I-contexts are transferred to the W-contexts of the associated Web service. These transfers are featured by a consolidation of these details before W-context is updates, as in (3, figure 5.6) and (4, figure 5.6).

*Reconsolidation during execution*. After an instance is executed, all the details from I-context are transferred to C-context. This is done after reconsolidation of the data to ensure the compatibility as there might be many different Web service providers, as in (3, figure 5.6) and (5, figure 5.6).

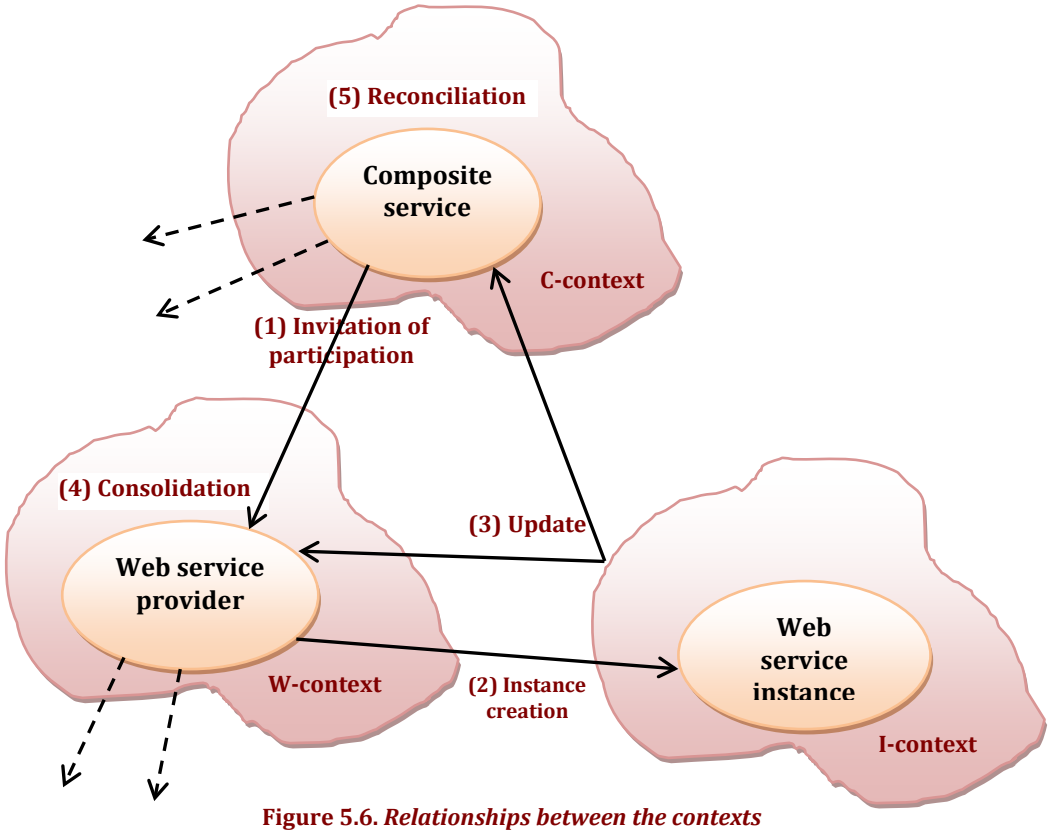



**Figure 5.6.** ***Relationships between the contexts***

The proposal of the paper [5] further develops ideas about that Web services are usually developed by different providers; therefore, talking about business processes, we have to talk about internal and external processes.

The service model that is proposed obtains operations of internal and external services and is expressed in the class diagram. So, on the one hand, common functionalities that are offered by services are identified in the class diagram, on the other hand, external services that are needed are imported into the model.

Both internal and external services have types. Now, we already established four types of message exchanging in order to execute a service operation transaction before, but it will not hurt to remember that once more. The table 5.10 summarizing these ideas are the following.

**Table 5.10.** ***Stereotypes of the service modelling***

| Stereotype | Description |
|---|---|
| <<own-service>> | Actual application service |

| | |
|---|---|
| **<<external-service>>** | Partner application service |
| **<<one-way-operation>>** | An asynchronous operation; client is the invoker and does not wait for a response |
| **<<request-response-operation>>** | A synchronous operation; client is the invoker and waits for a response from the server |
| **<<solicit-response-operation>>** | A synchronous operation; the server is the invoker and waits for a response from the client |
| **<<notification-operation>>** | An asynchronous operation; server is the invoker and does not wait for a response |

The meta model that goes with these ideas (and are called OOWS – Object Oriented Web Solution) is given in the following figure 5.7.

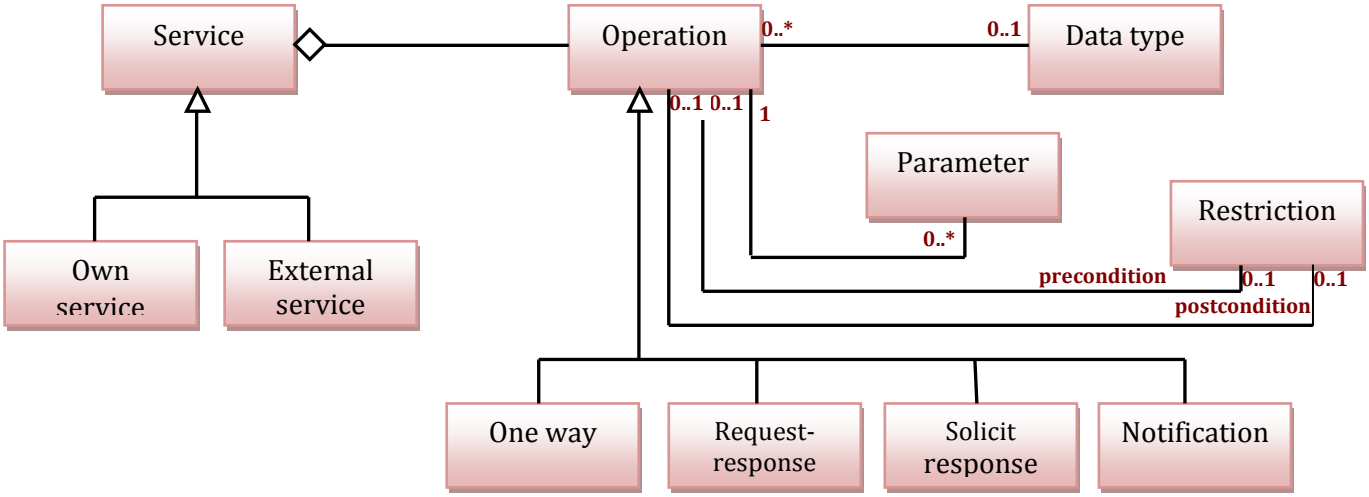



**Figure 5.7.** ***A meta model of the services***

This meta model simply shows that every service is one of two types, either it is 'own service' (internal) or 'external service'. A service is composed of operations. For that matter, operation has one of four types that are explained in the table above the model. What is more, operations can be defined by parameters, pre/post conditions and data types.

If a commerce Web system is designed to use services, the idea would be to make a notice about that in schemas modelling business processes. That includes all models that we already discussed in the part that talks about business modelling. That includes activity and workflow schemas. The idea is to put stereotypes expressing the type of the service. Stereotype <<own>> would go to services provided by the system itself, and <<external>> stereotype would represent services that are imported.

The ideas of transactions and services are very much related. Transactions are understood as a set of operations that has to be considered as a one unit: if executed – whole set is executed, if aborted – every operation of that set is aborted without any consequences.

Services (as a concept), on the one hand, can be understood as a single operation or as just the functionality of a particular system, on the other hand, Web services are considered as some functionality offered by an external provider. So that the services would be compatible, requirements should involve some agreed specifications and contexts which would allow services to interact with a system.

# 6. TAXONOMY

This part of the thesis presents the relationships of all the discussed models and summarizes ideas about the circumstances under which a particular method can be used. First of all, separate solutions for different aspects of modelling techniques are presented, and then a schema of relationships of all the given techniques is presented. Lastly, an example of how everything could be used in system's development process is suggested. To build all the schemas of the taxonomy we use our own methodology. The symbols used in the taxonomy are summarized in the following table 6.1.

**Table 6.1.** ***Symbols used in the taxonomy***

| Symbols | Explanation |
|---|---|
| **A square with rounded corners** | An element in the requirements process (an activity, a modelling technique, or an information unit) |
| **Curved arrows** | The sequencing (flow) of requirements processing |
| **Straight arrows** | Information contained by some certain processing technique |
| **A square with rounded corners and dotted texture** | An element in the requirements processing that was already discussed and given in some previous part of taxonomy |
| **A dashed arrow** | It indicates an indirect transition |
| **A dashed line** | Requires that the underlying logic of the methods would be compatible |

## 6.1. GENERAL REQUIREMENTS MODELLING

This part of taxonomy presents relationships within techniques of information modelling, navigation modelling, operation modelling, behaviour modelling, and presentation modelling.

The schema of taxonomy can be found in figure 6.1, description in table 6.2.

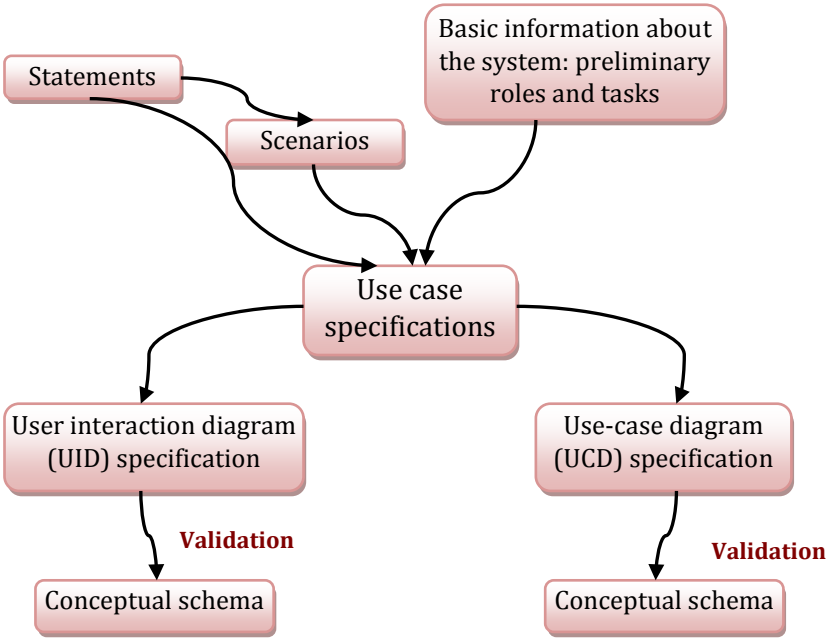



**Figure 6.1.** ***Taxonomy of information modelling***

**Table 6.2.** ***Description of information modelling taxonomy***

| Concepts | Descriptions |
|---|---|
| **Basic information** | First of all, it is important to gather some basic information about the system under development, including some preliminary information about:<br>*Roles* – indicating what kinds of actors are going to use the system;<br>*Tasks* – what kinds of activities the actors require from the system. |
| **Requirements gathering** | Although requirements could possibly be gathered during the whole process of system development, the main part of requirements gathering is done first thing before developing the system. This includes gathering collections of statements and scenarios.<br>*Statements* – collections of sentences expressing functional or non-functional expectations about the system under development.<br>*Scenarios* – collections of sentences formulating sequences of functional activities while using the system under development. |
| **UID** | User Interaction Diagrams are used mainly for expressing interactions between the user and the system. Exceptional features about this technology: the sequences of interactions can be easily shown, and information units and structures within interactions modelled. |
| **UCD** | Use-Case Diagrams are used for expressing separately different use cases within the system and showing relationships among them. Exceptional features about this technology: the diagram easily captures the expected functionality of the system with respect to different types of actors; what is more, different types of relationships among the functionality units are explicitly captured. |
| **Validation** | The act of requirements validation has to be made every step or every |

| | |
|---|---|
| | other step of requirements modelling. This is important as it costs so much to have errors in requirements – and requirements processing is used for eliminating those errors. The act of validation is an implicit one and will not be expressed again in the following schemas. |
| **Conceptual schema (based on UID)** | The conceptual schema (which usually is a class diagram) that is constructed based on UIDs very well captures informational units. That is, every interaction between the user and the system is oriented to one class diagram where the information related to a particular interaction is captured to attributes. What is more, there is a set of rules in the guidelines which helps to model the UIDs into conceptual schemas. |
| **Conceptual schema (based on UCD)** | The conceptual schema (which usually is a class diagram) that is constructed based on UCDs very well captures relationships between informational units. That is, every use case of the system is oriented to one class diagram where relationships from a use-case diagram are mapped to the conceptual schema. This technique does not capture attributes so well but it separates classes that are relevant for the system from those that are relevant for the user. |

The schema of taxonomy can be found in figure 6.2, description in table 6.3.

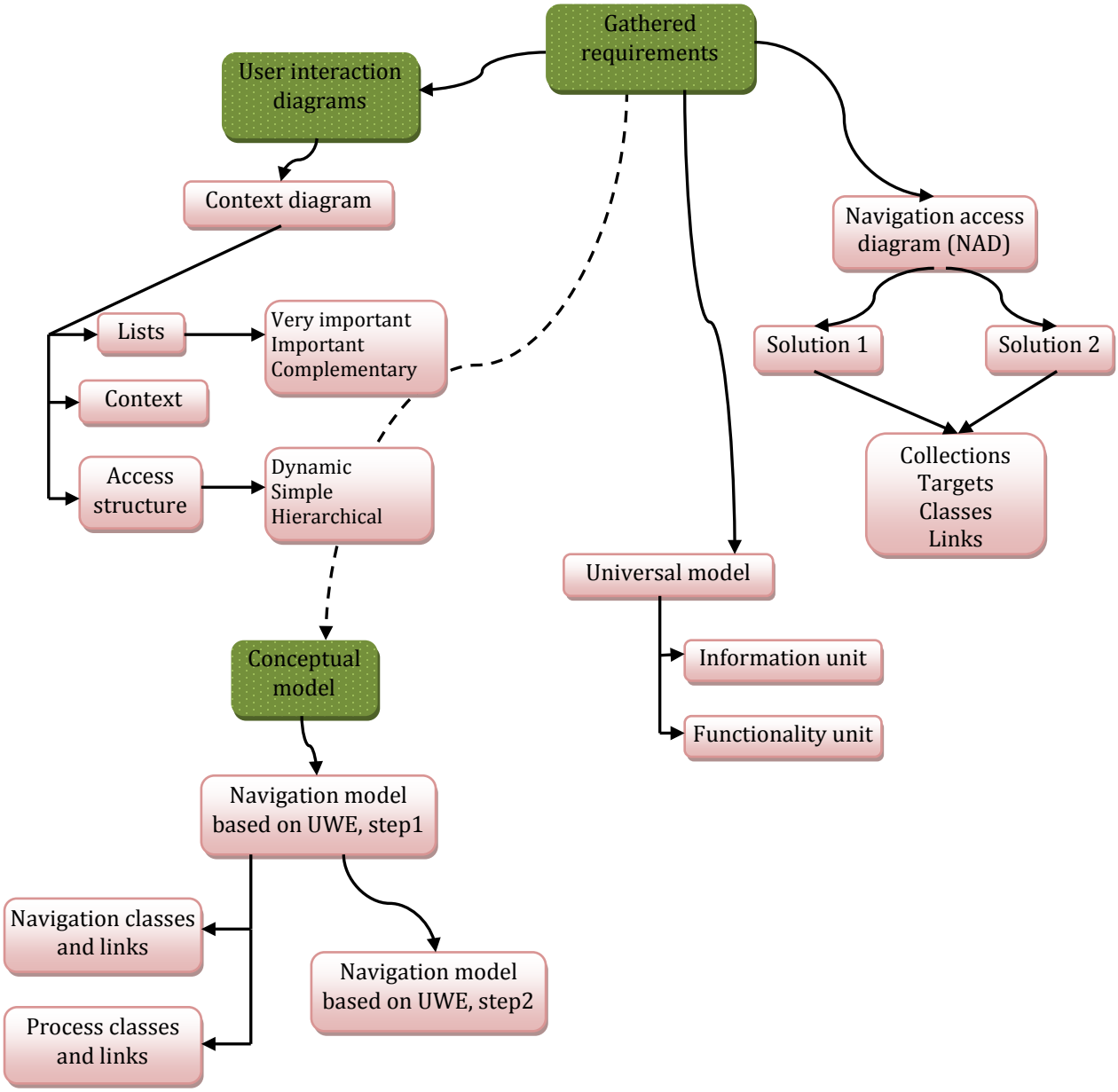



**Figure 6.2.** ***Taxonomy of navigation modelling***

**Table 6.3.** ***Description of navigation modelling taxonomy***

| Concepts | Descriptions |
|---|---|
| **Context diagram** | Context diagrams show navigational aspects of information units. When we have an information system and extract some information from it, it is important that we would get it in the right context. Technically, database queries are dealing with this aspect but that is out of scope of this thesis. The purpose of requirements processing is to model things so those |

| | |
|---|---|
| | queries would be written in the right way. There are three main concepts that are used to model context diagrams:<br>*Lists* – sets of similar units of information. Concerning lists, they have to be identified as holding very important information, important information, or complementary information. Navigation to reach different categories of information is modelled differently;<br>*Context* – a way to express relationships between objects and to show the exact dependencies;<br>*Access structure* – an element expressing actual navigation. Navigation in this way can be dynamic, simple, or hierarchical. |
| **NAD** | Navigation Access Diagrams are navigational models based on OO-H approach. The main aspects of these diagrams are:<br>*Collections* – capture the sense of menu, models the way for a user to access information;<br>*Targets* – considered to be some navigational subsystem fulfilling some navigational requirements;<br>*Classes* – views of actual domain (conceptual) classes;<br>*Links* – navigational paths in the system. |
| **NAD solution 1** | NAD solution 1 interprets OO-H in a way that a particular model shows how different navigation subsystems interact with classes from a conceptual schema, and the sequence of that interaction is shown as well. |
| **NAD solution 2** | NAD solution 2 interprets OO-H in a way that a particular model concentrates on applying relationships of classes and showing functionality within one navigation subsystem. One can model all the subsystems and put together in order to get the view of the whole system. |
| **Navigation model based on UWE, step 1** | Navigation model based on UWE is made in two steps. The first step gives a model of *navigational* and *process* classes. Navigational classes are mapped from conceptual schemas and process classes represent actual functionality within navigational classes. The relations are *navigational* between navigation classes and *process links* if the relationship is with a process class. |
| **Navigation model based on UWE, step 2** | The second step of this model extends the afore-given schema with elements representing access structures. The access structures can be *index*, *query*, or *guided tour*. |
| **Universal model** | The universal model (as, for example, the first step of navigation model based on UWE also proposes) suggests capturing in one schema information and functionality. This particular method suggests capturing these aspects with respect to some certain information unit. This gives a better understanding of the system from a certain point of view and let's minimizing mistakes. These are the most important parts:<br>*Information unit* – a view of attributes and operations of a class in the conceptual schema;<br>*Functionality unit* – captures own or external functionality when the user is navigating. |

The schema of taxonomy can be found in figure 6.3, description in table 6.4.

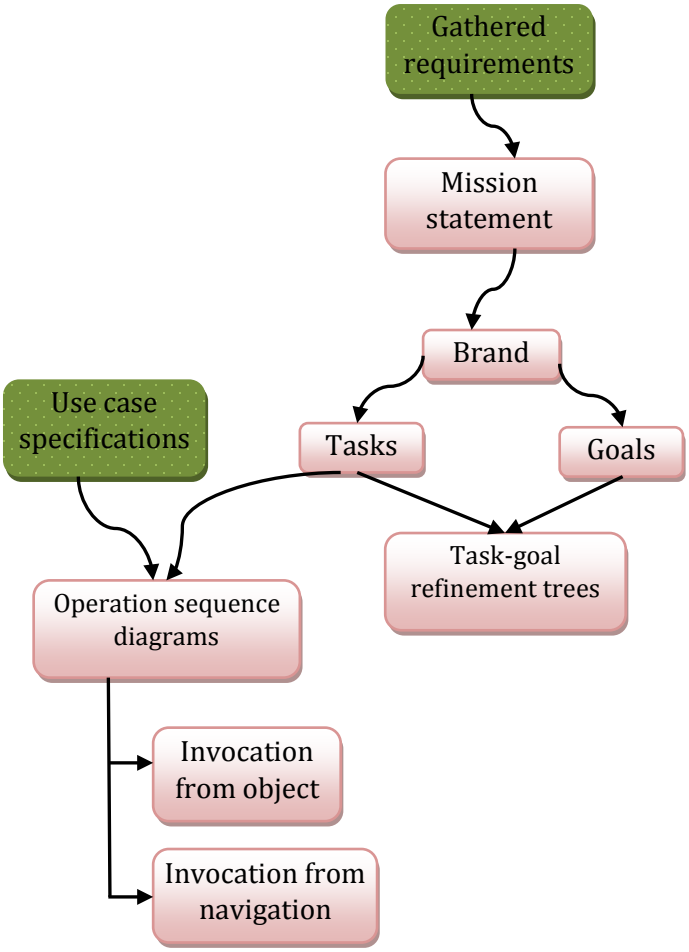



**Figure 6.3.** ***Taxonomy of operation modelling***

**Table 6.4.** ***Description of operation modelling taxonomy***

| Concepts | Descriptions |
|---|---|
| **Mission statement** | It is a general description of a system: what it's all about, expected content, functionality, and context. |
| **Brand** | It sums information about the provider (describing what kind of content the system offers) and the user (describing actions which indicate the functionality of the system). |
| **Tasks** | Short descriptions of what a system has to do with respect to a particular user. |
| **Goals** | These are short descriptions of some end-state in the system with a relationship to a particular task. |
| **Refinement trees** | These are modelled requirements of the system in a way that all the tasks would be covered together with information of the roles that can perform those tasks. What is more, expectations are modelled towards the system by defining goals - not just for a system but for all functional aspects of it. |

| | |
|---|---|
| **Operation sequence diagram** | Operation sequence diagram is based on the UML sequence diagrams but with relevant adjustments. It shows the sequencing of operation execution within the system together with navigational sense from user's point of view. There are two important concepts within this methodology:<br>*Invocation from object* – operations can be invoked from an object of a given type (as entities, collections, and associations).<br>*Invocation from navigation* – operations can be invoked from a certain navigational context, or from objects reached within some constraints. |

The schema of taxonomy can be found in figure 6.4, description in table 6.5.

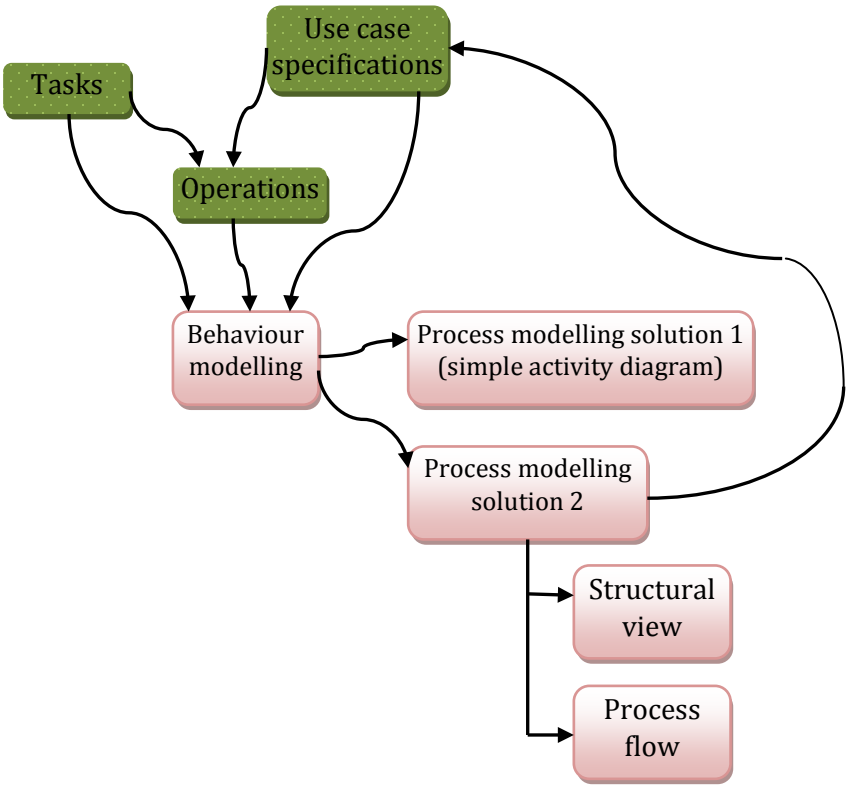



**Figure 6.4.** ***Taxonomy of behaviour modelling***

**Table 6.5.** ***Description of behaviour modelling taxonomy***

| Concepts | Descriptions |
|---|---|
| **Process model solution 1** | The first solution for process modelling is based on UML activity diagrams. It captures the activity process from a user's point of view and is oriented to use cases which we have already discussed before. |
| **Process model solution 2** | The second solution of process modelling is based on UWE approach. The model is made in two steps: the first step gives a structural view of the system and the second step presents a process flow. |
| **Structural view** | It is all about capturing process related information with respect to structure and behaviour. It is similar in some sense to class diagrams. |
| **Process flow** | The process flow schema is modelled by matching the structural view into the UML's activity diagram. In this way not only the functionality aspect is captured but relations with certain informational units as well. |

The schema of taxonomy can be found in figure 6.5, description in table 6.6.

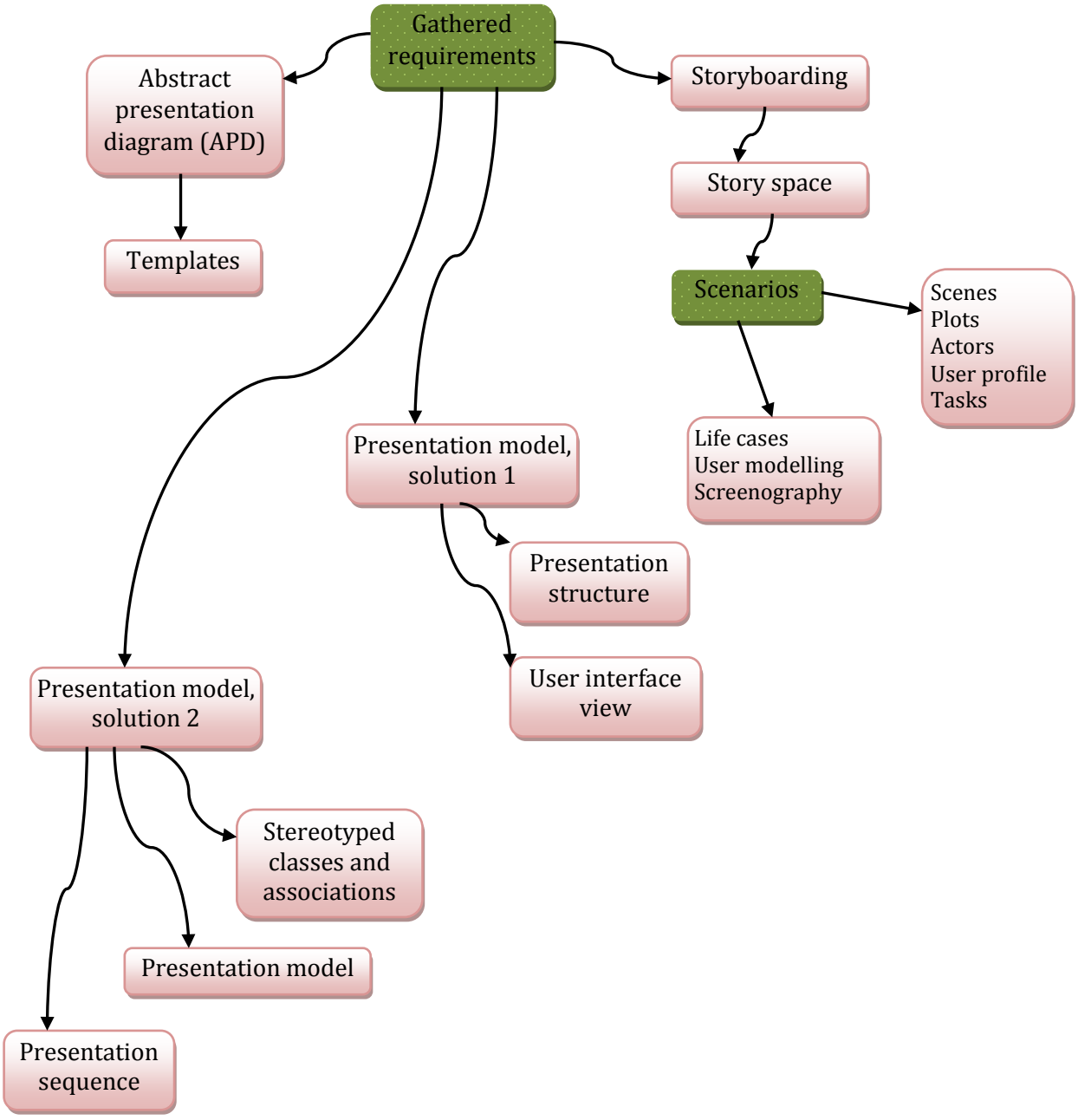



**Figure 6.5.** ***Taxonomy of presentation modelling***

**Table 6.6.** ***Description of presentation modelling taxonomy***

| Concepts | Descriptions |
|---|---|
| **APD** | The idea of an Abstract Presentation Diagram is borrowed from OO-H methodology. There are five main aspects of presentation captured in this kind of model. These aspects are given in a form of templates: *information structures*, *style features*, *forms*, *functionality*, and *windows*. |

| | |
|---|---|
| **Storyboarding** | It is a description of the system by giving details about story spaces and associated scenarios. |
| **Story spaces** | It is a space (usually with respect to a certain type of a user) within which some set of scenarios takes place. |
| **Scenarios** | As we have already talked, scenarios are collections of sentences formulating sequences of functional activities while using the system under development. Related concepts are: *scenes*, *plots*, *actors*, *user profiles*, and *tasks*. |
| **Life cases** | These are observations of different aspects of user's skills and experiences, significance of time and place, etc. |
| **User modelling** | It is a specification of user profiles and portfolios in order to recognize which tasks are relevant to which roles. |
| **Screenography** | This technique is responsible for summarizing all the relevant information with respect to a certain user. It aims at an individualized, decorated playout in consideration of user profiles and portfolios, provides aims, context, equipment and storyline process. |
| **Presentation model, solution 1** | This presentation model is based on UWE. The idea is to build a physical representation on the logic model. This model comes with two different schemas: a *presentation structure* and a *user interface view*. |
| **Presentation structure** | Presentation structure tries to capture the system in three different levels: the *central location* of structuring, *presentation alternatives*, and *logical page fragments*. |
| **User interface view** | User interface view actually captures the presentation of one actual Web page. For this matter, all sorts of stereotypes can be used. In order to model the whole system, user interface view of every single page has to be given. |
| **Presentation model, solution 2** | This second presentation model analyzes the windows and parts of them together with relationships. For this model stereotyped classes and associations are exceptional. What is more, this technique comes with two types of schemas: a presentation model, and a presentation sequence diagram. |
| **Stereotyped classes and associations** | *Stereotyped classes* capture aspects as windows, frames, what is more, client and server dependencies.<br>*Stereotyped associations* capture aspects as submit, link, display, etc. |
| **Presentation model** | A presentation model works with elements representing aforementioned stereotypes. The idea is to exactly show how windows and parts of them are related in an actual system. |
| **Presentation sequence diagram** | As a complement to presentation model, a sequence diagram captures the sequence of element display with respect to user's navigation or operation execution. |

## 6.2. BUSINESS AND SERVICE REQUIREMENTS MODELLING

This part of taxonomy presents relationships within techniques of business modelling, transaction modelling, and service modelling. Transaction and service modelling are given in the same schema.

### BUSINESS MODELLING

The schema of taxonomy can be found in figure 6.6, description in table 6.7.

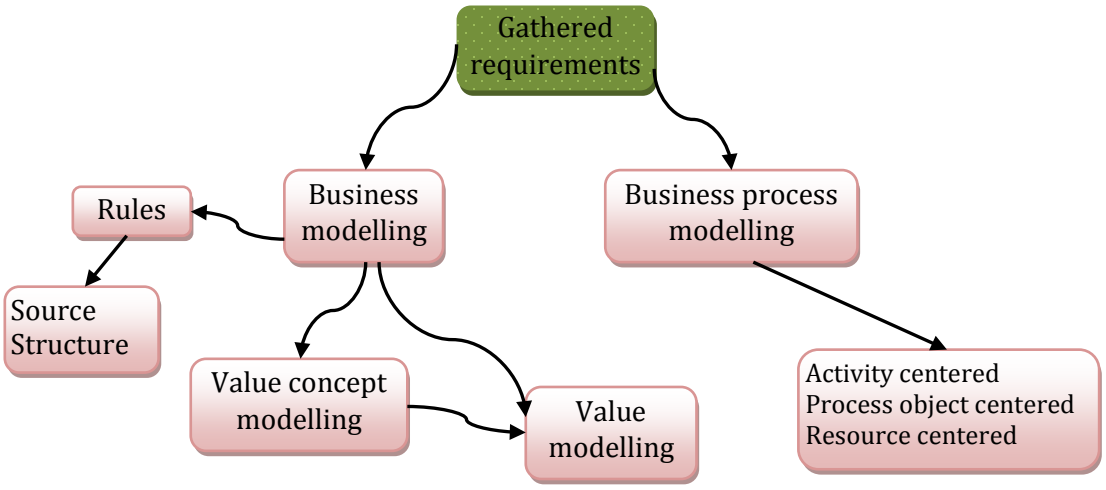



**Figure 6.6.** ***Taxonomy of business modelling***

**Table 6.7.** ***Description of business modelling taxonomy***

| Concepts | Descriptions |
|---|---|
| **Business process modelling** | Business process models show the way the value is exchanged within the system. It includes setting a common approach for the work, process supported by systems, property analysis, etc. When modelling, processes that are modelled can be of three different types: *activity centered*, *process object centered*, and *resource centered*. |
| **Business modelling** | Business modelling provides the design grounds for a system from a business point of view. The idea is to identify what is exchanged and who are the participants making that exchange. |
| **Rules** | There are some rules (information) to have in mind when modelling a business. Classified according to the *source* we have: *mandates*, *policies*, and *guidelines*. Classified according to the *structure* we have: *integrity*, *derivation*, *reaction*, *production*, and *transformation*.<br>What is more, concepts as *lawful event space* and *conceivable state space* might come in handy. |
| **Value concept modelling** | While modelling value, these are the concepts to be taken into consideration: *value offering*, *value exchange*, *value activity*, *value* |

| | |
|---|---|
| **Value modelling** | *interface*, *value port*, and *value object*. All these concepts are related and could be put into a schema to show that. The relations could be shown for general concepts, as well as for specific values. |
| | Actual value model in the particular system is captured in some sort of activity diagrams. The idea is to use swimlanes for every kind of an actor that is participating in the business and, by having activities as value adding activities, to show the flow of the value that is exchanged. |

## TRANSACTION AND SERVICE MODELLING

The schema of taxonomy can be found in figure 6.7, description in table 6.8.

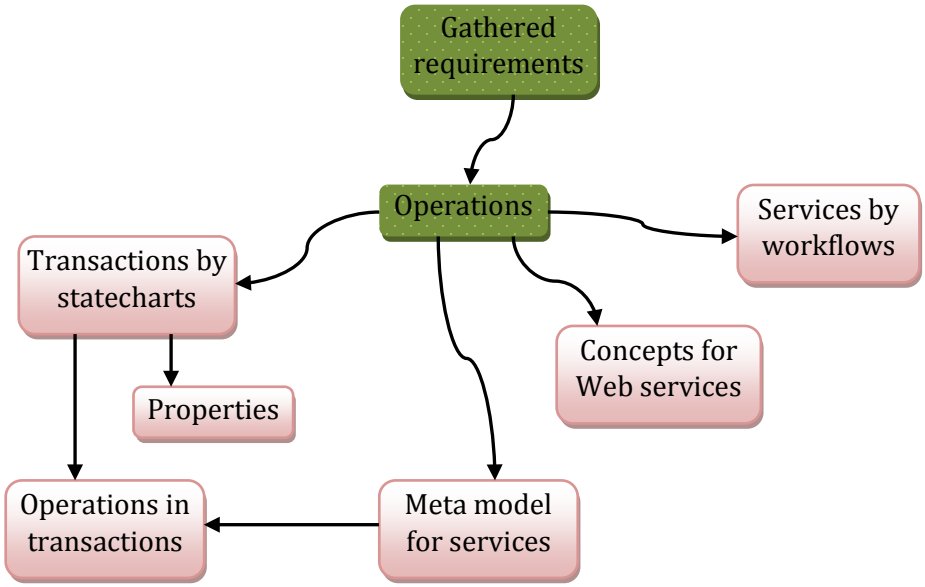



**Figure 6.7.** ***Taxonomy of transaction and service modelling***

**Table 6.8.** ***Description of transaction and service modelling taxonomy***

| Concepts | Descriptions |
|---|---|
| **Transactions by statecharts** | Statecharts could be a good tool to express transactions in a sense that the transaction itself can be defined by being in states, as simple as in: *start*, *cancel*, or *end*. |
| **Operations in transaction** | Operations in transactions (the same as for services) could be of four types: one message exchange (*one-way*, *notification*) and two message exchange (*request-response*, *solicit-response*). |
| **Properties** | Transactions (with respect to Web services) have some types of information to be discussed: *implicit and timed transitions*, *compensation*, *resource locking*, *conditions* and *instance-specific properties*, and *multi-state enabled operations*. |
| **Services by workflows** | Workflows let showing how services are executed form the start to the end. The flow of executions is dependable upon individual choices of the user. What is more, aspects of synchronizations can be captured as well. |

| | |
|---|---|
| **Meta model for services** | The meta model for services captures information about operations in services and other relevant ideas. The most important thing is that a distinction is done between what is an *own* service and what is an *external* service. |
| **Context for Web services** | Modelling contexts of Web services is important because Web services usually are given by an external provider and the challenge is to make them compatible with the system under development. |

This part of the thesis presents taxonomy of all the techniques that we talked about together. Previously we explained how methods are related within different aspects of modelling, accordingly, this particular taxonomy captures the full view. The schema given in figure 6.8 has been simplified in order to express the most relevant ideas. Arrows show the relationship flow, whereas dashed lines indicate that given information in different techniques has to be compatible. All the dependencies are coloured so the taxonomy would be the least confusing as possible. This is just a variant of the dependencies among the given techniques that might exist; nevertheless, it represents relationships accordingly to what all the guidelines given in the thesis suggest.

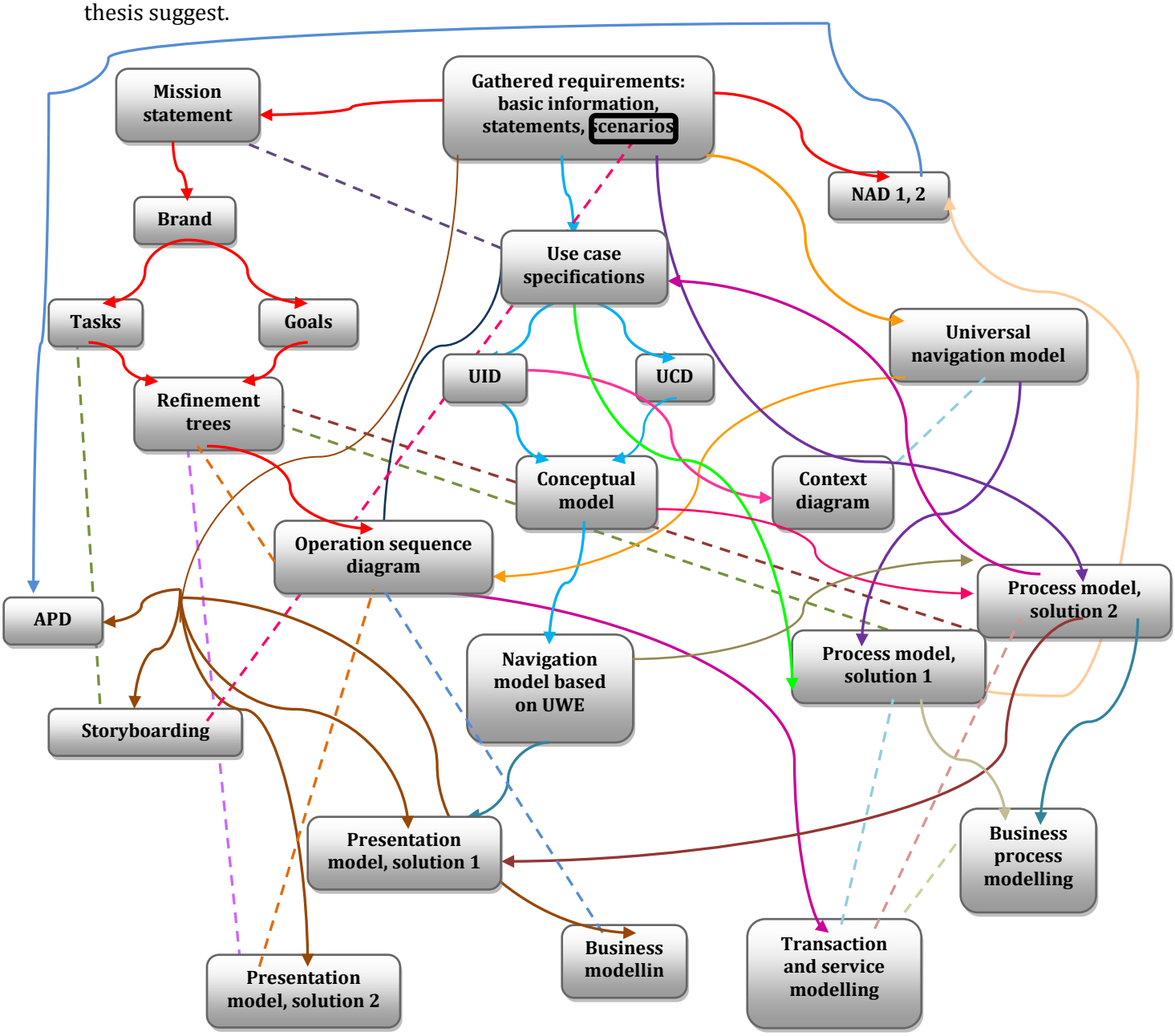



Figure 6.8. *Taxonomy of all the techniques together*

In the beginning of the thesis we had an example presenting abstraction layers to a Web based information system. We had that because in order to develop such a system, those layers have to be dealt with: *strategic layer*, *business layer*, *conceptual layer*, *presentation layer*, and *implementation layer*. This is one way of many to deal with Web based information system creation in a structured fashion. Every of these layers need different aspects of requirements processing. The table 6.9 below gives an example of a solution for this particular structure.

**Table 6.9.** ***An example to the solution***

| Layer | Processed requirements |
|---|---|
| **Strategic layer** | Mission statement;<br>Statements and scenarios; |
| **Business layer** | Storyboarding;<br>Task-goal refinement trees;<br>Value model; |
| **Conceptual layer** | UCD based conceptual schema;<br>Process model, solution 2;<br>Context diagram;<br>NAD; |
| **Presentation layer** | Presentation model, solution 2;<br>(operation sequence?); |
| **Implementation layer** | Transactions by statecharts;<br>Contexts for services. |

This solution is given assuming that we are modelling a Web system similar to the one about photo purchasing. This is a commercial system; therefore, value and process models are important as an aspect of business modelling. The concentration point is not the exact information that a user gets from a system so much as what the user can do in the system, therefore conceptual schema is build with respect to use cases, but the context diagram assures that there are no mistakes in queries. The layout structure of presentation solution 2 is chosen because this model can capture a sense of navigation as well. In this way it is easy to see how the presentation changes with respect to some operation execution. What is more, in this case requirements processing can come and into implementation layer as we use transactions and an external payment system which has to be compatible with the one we are creating. Finally, if there appears to be some complicated situations with functionality, we leave ourselves an option to model sequences of operations.

While the example seems to work for our benefit, let's make sure that we cover all the areas with respect to data, functions, views, and dialogues. We had this in mind when talking about abstraction layers in 3.2 part of the thesis.

**Table 6.10.** ***Dimensions of modelling techniques***

| Modus\Focus | Global | Local |
|---|---|---|
| **Static** | **Data specification:**<br>*Storyboarding;*<br>*Task-goal refinement tree;*<br>*Value modelling;* | **View specification:**<br>*Presentation model, solution 2;*<br>*NAD;* |

| | | |
|---|---|---|
| **Dynamic** | *UCD based conceptual schema;* | |
| | **Function specification:**<br>*Task-goal refinement tree;*<br>*Process model, solution 2;*<br>*Context diagram;* | **Dialogue specification:**<br>*NAD;*<br>*Operation sequence;*<br>*Transactions by statecharts;*<br>*Contexts for services;* |

As it can be seen from the table, we cover all the necessary aspects. Nevertheless, it is easier to separate methods with respect to weather they are local or global. The distinction between weather they are static or dynamic is not so obvious. Sometimes one technique can contain both features.

## 6.5. DISCUSSION

When modelling a Web based information system, whenever there is a doubt about which techniques to use in general or which kinds of modelling aspects to choose, the given taxonomy comes in handy. It presents not only all the possibilities within a certain aspect of modelling, but also gives descriptions by explaining the differences. What is more, it presents taxonomy of all the possible different techniques together so it would be clear how methods are related and what is the logical sequence when processing requirements in the chosen way.

What is more, there is an example of the suggested development structurization that is given with the solution taken from the discussed taxonomy. It is worth mentioning once again, that the subset of chosen techniques is entirely dependable upon what kind of the system under development we are talking about and what it suppose to do.

# CONCLUSION

This thesis talks about requirements processing in Web based information systems by presenting guidelines for a set of relevant methods and by supporting ideas with taxonomy. Overall, the thesis contains six main parts: types of requirements, ideas behind requirements, an overview towards design approaches, guidelines for general requirements models, guidelines for more specific requirements models, and taxonomy.

The 'types of requirements' part pursuits the idea that there exist functional and non-functional requirements when it comes to Web system modelling. And as non-functional requirements are basically concerned with the entire hierarchy of the system, functional requirements are the ones that can be processed in advance so that everything would be known about the system's functionality before it's being started to be build. Nevertheless, non-functional requirements must be had in mind along the way. Furthermore, 'ideas behind requirements' present a rationale for this entire thesis by convincing that it is very important to find out the real user requirements by processing them before (sometimes while) the work on the system is started. This is the way to minimize the mistakes, and thus minimize the cost and time spent. Modelling as a requirements processing technique is one of the best when it comes to Web based information systems. In addition to that, an 'overview towards design approaches' gives some sense of what to expect further in the thesis: what approaches with respect to Web systems are discussed.

Next thing that is given are guidelines for general requirements models. The meaning of *general* is that they are the most common approaches when it comes to Web based systems. We discuss and give guidelines for information modelling (2 techniques), navigation modelling (5 techniques), operation modelling (2 techniques), behaviour modelling (2 techniques), and presentation modelling (4 techniques). What is more, there are more to Web based systems when it comes to commercial and service systems. Then concepts as business modelling, transactions, and services are relevant. For this matter, a part for specific cases of requirements is given. First of all, there are some ideas of what to do if we are building a system for an entire enterprise. Moreover, how to separate thoughts of what is business modelling and what is business process modelling. In addition, there are given two ideas of how to model the value. Furthermore, there is a suggestion given towards transaction modelling and three suggestions towards service modelling.

Finally, the last part of the thesis gives taxonomy of all the techniques that were discussed. First of all, some thorough schemas with descriptions are given for different approaches of requirements processing, such as information modelling, navigation modelling, etc. After that, there is a schema presenting relationships and flows of dependability of all the techniques together. It is a summarized version, as to capture all the details would be too confusing. And so it would be more clear what one could do with all of that, a small practical example is suggested that explains how the taxonomy could be used.

When one wants to develop a Web based system and if he or she is not so much trained, there is a big question where to start when one has just some requirements on hand. And this thesis gives an answer to this question. It helps to understand what the needed requirements are and

proposes many solutions towards how they can be processed. The detailed taxonomies give ideas about different approaches, and the abstract taxonomy produces the path among the techniques selected, whereas guidelines explain how to do the actual work by supporting ideas with simple examples.

# REFERENCES


[1]. Güell, N., Schwabe, D., and Vilain, P. 2000. Modeling Interactions and Navigation in Web Applications. In *Proceedings of the Workshops on Conceptual Modeling Approaches For E-Business and the World Wide Web and Conceptual Modeling: Conceptual Modeling For E-Business and the Web* (October 09 - 12, 2000). S. W. Liddle, H. C. Mayr, and B. Thalheim, Eds. Lecture Notes In Computer Science, vol. 1921. Springer-Verlag, London, 115-127.

[2]. Nora Koch, Andreas Kraus, Cristina Cachero and Santiago Melia. Modeling Web Business Processes with OO-H and UWE. Third Int. Worskhop on Web-oriented Software Technology (IWWOST'03), 2003.

[3]. L. Baresi, F. Garzotto and P. Paolini, "From Web Sites to Web Applications: New Issues for Conceptual Modeling," *Conceptual Modeling for E-business and the Web,* Lecture Notes in Computer Science 1921, Springer-Verlag, Berlin, 2000

[4]. Gómez, J., Cachero, C., and Pastor, O. 2001. Conceptual Modeling of Device-Independent Web Applications. *IEEE MultiMedia* 8, 2 (Apr. 2001), 26-39.

[5]. Quintero, R., Torres, V., Ruiz, M., and Pelechano, V. 2006. A conceptual modeling approach for the design of web applications based on services. In *Proceedings of the 44th Annual Southeast Regional Conference* (Melbourne, Florida, March 10 - 12, 2006). ACM-SE 44. ACM, New York, NY, 464-469.

[6]. Ceri, S., Fraternali, P., and Matera, M. 2002. Conceptual Modeling of Data-Intensive Web Applications. *IEEE Internet Computing* 6, 4 (Jul. 2002), 20-30.

[7]. Schewe, K., Zhao, J., and Thalheim, B. 2007. Quality Assurance in Web Information Systems Development. In *Proceedings of the Seventh international Conference on Quality Software* (October 11 - 12, 2007). QSIC. IEEE Computer Society, Washington, DC, 219-224.

[8]. Distante, D. and Tilley, S.: “Conceptual Modeling of Web Application Transactions: Towards a Revised and Extended Version of the UWA Transaction Design Model”. *In Proceedings of the 11th International Multi-Media Modelling Conference* (MMM 2005: Jan. 12-14, 2005; Melbourne, Australia). Los Alamitos, CA: IEEE Computer Society Press, 2005.

[9]. Gordijn, J., Akkermans, H., and Vliet, H. v. 2000. Business Modelling Is Not Process Modelling. In *Proceedings of the Workshops on Conceptual Modeling Approaches For E-Business and the World Wide Web and Conceptual Modeling: Conceptual Modeling For E-Business and the Web* (October 09 - 12, 2000). S. W. Liddle, H. C. Mayr, and B. Thalheim, Eds. Lecture Notes In Computer Science, vol. 1921. Springer-Verlag, London, 40-51.

[10]. Frank, U. 2002. Multi-perspective Enterprise Modeling (MEMO) - Conceptual Framework and Modeling Languages. In *Proceedings of the 35th Annual Hawaii*

*international Conference on System Sciences (Hicss'02)-Volume 3 - Volume 3* (January 07 - 10, 2002). HICSS. IEEE Computer Society, Washington, DC, 72. 22

[11]. Schewe, K. and Thalheim, B. 2005. Conceptual modelling of web information systems. *Data Knowl. Eng.* 54, 2 (Aug. 2005) , 147-188.

[12]. Young, R. R. 2000 *Effective Requirements Practices*. Addison-Wesley Longman Publishing Co., Inc.

[13]. Wiegers, K. E. 2003 *Software Requirements*. 2. Microsoft Press.

[14]. zur Muehlen, M., Indulska, M., and Kamp, G. 2007. Business process and business rule modeling languages for compliance management: a representational analysis. In *Tutorials, Posters, Panels and industrial Contributions At the 26th international Conference on Conceptual Modeling - Volume 83* (Auckland, New Zealand, November 01 - 01, 2007). J. Grundy, S. Hartmann, A. H. Laender, L. Maciaszek, and J. F. Roddick, Eds. ACM International Conference Proceeding Series, vol. 334. Australian Computer Society, Darlinghurst, Australia, 127-132.

[15]. Boualem Benatallah, Fabio Casati, Farouk Toumani, Rachid Hamadi, Conceptual Modeling of Web Service Conversations, pp. 449-467, Proceedings of the 15th Conference on Advanced Information Systems Engineering, Johann Eder, Michele Missikoff (Ed.), Lecture Notes in Computer Science, Springer-Verlag, Klagenfurt, Austria, Lecture Notes in Computer Science, Vol. 2681, June 2003.

[16]. Unified Modeling Language User Guide, The (2nd Edition) (The Addison-Wesley Object Technology Series) by Grady Booch, James Rumbaugh, Ivar Jacobson. Publisher: Addison-Wesley Professional; 2 edition (May 19, 2005)

[17]. Cachero, C., Gómez, J., and Pastor, O. 2000. Object-Oriented Conceptual Modeling of Web Application Interfaces: the OO-HMethod Abstract Presentation Model. In *Proceedings of the First international Conference on Electronic Commerce and Web Technologies* (September 04 - 06, 2000). K. Bauknecht, S. K. Madria, and G. Pernul, Eds. Lecture Notes In Computer Science, vol. 1875. Springer-Verlag, London, 206-215.

[18]. Narendra, N. C. and Orriens, B. 2007. Modeling web service composition and execution via a requirements-driven approach. In *Proceedings of the 2007 ACM Symposium on Applied Computing* (Seoul, Korea, March 11 - 15, 2007). SAC '07. ACM, New York, NY, 1642-1648.

[19]. R. Kazman, M. Klein, P. Clements, ATAM: Method for architecture evaluation, Technical Report, SEI, CMU/SEI-2000-TR-004, ESC-TR-2000-004, August 2000.

[20]. *CS2 Software Engineering note2,* CS2Ah Autumn 2004, The University of Edinburg, School of Informatics based on those provided by Ian Sommerville

[21]. *Functional Requirements and Use Cases*, by Ruth Malan and Dana Bredemeyer, June 1999.

[22]. *Defining Non-Functional Requirements or System Qualities*, by Ruth Malan and Dana Bredemeyer, August 2001.

[23]. Quintero, R., Torres, V., Ruiz, M., and Pelechano, V. 2006. A conceptual modeling approach for the design of web applications based on services. In *Proceedings of the 44th Annual Southeast Regional Conference* (Melbourne, Florida, March 10 - 12, 2006). ACM-SE 44. ACM, New York, NY, 464-469.

[24]. Hennicker, R. and Koch, N. 2001. Modeling the User Interface of Web Applications with UML. In *Workshop of the Puml-Group Held Together with the «Uml»2001 on Practical Uml-Based Rigorous Development Methods - Countering Or integrating the Extremists* A. Evans, R. B. France, A. M. Moreira, and B. Rumpe, Eds. LNI, vol. 7. GI, 158-172.

[25]. Benatallah, B., Casati, F., and Toumani, F. 2004. Web Service Conversation Modeling: A Cornerstone for E-Business Automation. *IEEE Internet Computing* 8, 1 (Jan. 2004), 46-54.

# APPENDIX A

The process of requirements given by R. Young in [12] addresses these activities:

- Identifying requirements
- Understanding the customer's needs
- Clarifying and restating the requirements
- Analyzing the requirements
- Defining the requirements
- Specifying the requirements
- Prioritizing the requirements
- Deriving requirements
- Partitioning requirements
- Allocating requirements
- Tracking requirements
- Managing requirements
- Testing and verifying requirements
- Validating requirements

The alternative techniques in order to reveal real customer requirements given by R. Young in [12] include following:

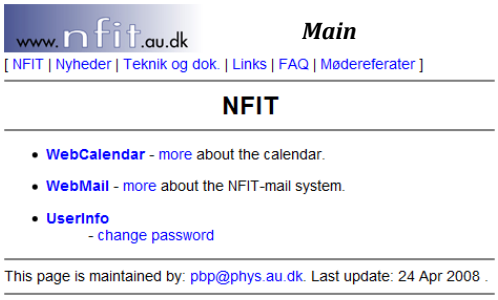
- Rapid application development
- Object-oriented methods
- Joint application design
- Use cases
- Market surveys
- Questionnaires
- Customer and user interviews
- Limited capability/rapid prototyping
- Modelling
- Finite-state machines
- State transition diagrams
- User-defined operational scenarios
- Process management
- Joint team meetings
- "Rules of conduct"
- Requirements elicitation methods
- Brainstorming and multivoting
- Quality function deployment
- Interface control working groups
- Technical interchange meetings
- Operational scenarios obtained from users

- Prototyping
- Beta testing
- Observation of existing systems, environments, and workflow patterns
- MOEs
- Trade studies
- Mathematical techniques (design of experiments, sensitivity analysis, timing, sizing, Monte Carlo simulation)
- Requirements validation
- System capability concept
- Formal program reviews
- In-process reviews
- Status meetings
- Teleconferences
- Focus groups
- Storyboards

The sources of requirements changes given by R. Young in [12] include:

- The expanded quantity of requirements associated with highly complex systems. Because of the large scope of the system, less rigor is applied to providing essential system function.
- The increased interaction and information sharing within and between systems.
- Poorly or loosely defined requirements from the onset (including failure to discover and to emerge the real requirements).
- Changes in business objectives and plans.
- Technology changes.
- Changes in law, policies, directives, and so forth.
- Customers and users who change their minds about things or learn of 'neat' ways to do things.
- Developers who add their own special twists or even make requirements decisions.

As an example to help modelling different approaches of requirements processing techniques we use *http://www.nfit.au.dk/* as our 'suppose to be' system under development. Here will be presented the main aspects of this Web system that are related to the discussed approaches.




***Main***

The purpose of this system is basically to provide some relevant information and to allow access to the mail. From the main page one can access the mail server: following the 'WebMail' link.

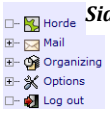

***Log-in***

The link gives the main page of the mail server. There one can produce log in details (username and password) and in this way to access the account of the mail server. When logged in, there are three parts of the Web page:

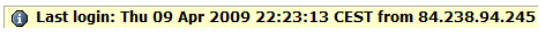
***Header***

the header with functionality icons, the side-bar with functionality icons, and the main part of the page where all the work is done (in the example we have last registered IP information displayed).

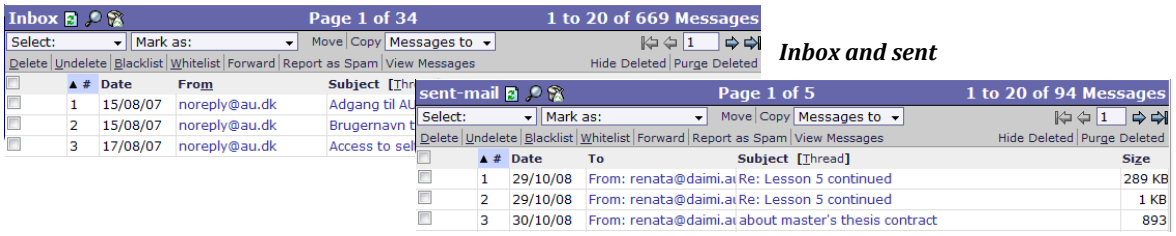
***Side-bar***




***Main part***




When presenting models, we talk about inbox and sent-mail.




***Inbox and sent***

What is more, information and functionality aspects are also discussed about e-mails and forms for writing e-mails.

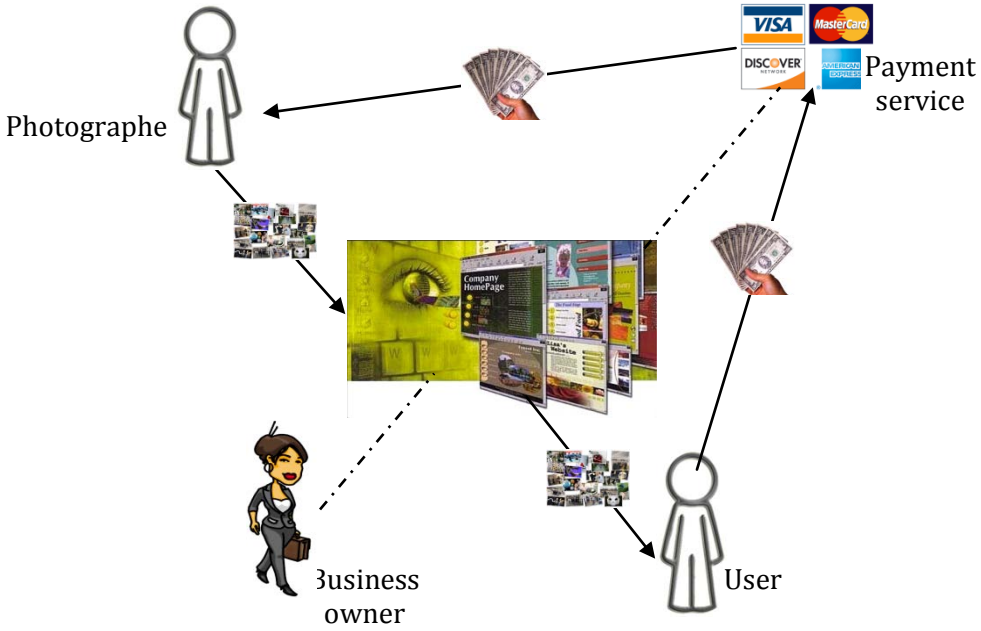
***E-mail***


***E-mail form***

The models that are given about this system (within information, navigation, operation, behaviour, and presentation requirements processing guidelines) are not entirely accurate to this existing system. All the adjustments are done in order to present as many capabilities of the methods as possible. After all, this thesis is all about showing what one can exactly do when using a certain modelling methodology.

# APPENDIX C

When talking about business systems and services, the example of the Web mail system presented in appendix B no longer is so expressive to fit the methodologies. For that matter, we thought of some sort of an abstract commercial system. There are just basic ideas about a site providing pictures that users can buy. An amount of information given is just necessary in order to show in relevant models.




What this kind of a system is beneficial for our examples and what the afore-discussed system (Web mail) lacks is first of all exchange of an actual value. What is more, a payment service is an external provider of Web services, and finally, as we are talking about money, the terms of transactions and services are especially relevant.